



**MATEMATICKO-FYZIKÁLNÍ
FAKULTA**
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Jakub Levý

Využití Puppeteeru pro automatizaci akcí webového prohlížeče

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Jakub Klímek, Ph.D.

Studijní program: Informatika

Studijní obor: Obecná informatika

Praha 2021

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V _____ dne _____

Podpis autora

Poděkování.

Název práce: Využití Puppeteeru pro automatizaci akcí webového prohlížeče

Autor: Jakub Levý

Katedra: Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Jakub Klímek, Ph.D.

Abstrakt: Abstrakt.

Klíčová slova: Puppeteer, automatizace prohlížeče

Title: Usage of Puppeteer for automation of web browser actions

Author: Jakub Levý

Department: Department of Software Engineering

Supervisor: RNDr. Jakub Klímek, Ph.D.

Abstract: Abstract.

Keywords: Puppeteer, web browser automation

Obsah

Seznam nestandardních a méně známých zkratk	3
1 Úvod	4
1.1 Cíle práce	4
1.2 Struktura textu	4
1.3 Konvence	4
1.4 Využitý software a technologie	4
2 Seznámení s technologiemi k automatizaci prohlížečů	5
2.1 Selenium	5
2.2 Puppeteer	8
2.3 Playwright	10
2.4 PhantomJS	13
3 Nahrávání a přehrávání akcí	15
3.1 Konkurenční řešení	15
3.1.1 Selenium IDE	15
3.1.2 Katalon Recorder	20
3.1.3 Ranorex Recorder	21
3.1.4 Headless Recorder (dříve Puppeteer Recorder)	22
3.2 Využití Puppeteeru	25
3.2.1 WindowEvents	25
3.2.2 ViewportEvents	27
4 Řešení	33
4.1 Design	33
4.2 Scénáře použití	36
4.3 Porovnání funkcionality oproti konkurenčním řešením	38
4.4 Tutoriály	44
4.4.1 Prvotní spuštění	44
4.4.2 Nová nahrávka	45
4.4.3 Připojení ke vzdálenému Chromu	52
4.4.4 Tlačítko JSON	54
4.4.5 Výběr akcí ke zpracování	55
4.4.6 Filtř akcí	57
4.4.7 Změna identifikátoru akce	60
4.5 Detaily implementace	62
4.5.1 Frontend	62
4.5.2 Backend	62

5 Závěr	63
A Fragmenty kódu pro Puppeteer	64
A.1 Zachycení událost z viewportu	64
A.2 Momentálně aktivní tab	65
A.3 Oprávnění „push“	65
A.4 Element podle lokátoru	66
A.5 Práce se schránkou	67
A.6 Pokus o získání oprávnění pro práci se schránkou	68
A.7 Čtení ze schránky s CDP oprávněním „clipboardReadWrite“	69
A.8 Úspěšné čtení ze schránky i zápis do schránky	70
A.9 Využití NetMQ a ZeroMQ.js	70
Slovníček vybraných pojmů	71
Seznam obrázků	75
Seznam tabulek	75
Seznam ukázek kódu	76
Reference	77

Seznam nestandardních a méně známých zkratk

CDP Chrome DevTools Protocol

Chrome Chromium/Google Chrome/Microsoft Edge

FRP Firefox Remote Protocol

Jazyk Programovací jazyk

Prohlížeč Webový prohlížeč

Selektor CSS Selektor

Stránka Webová stránka

1 Úvod

Upozorňujeme čtenáře, že na konci této práce se nachází Slovníček vybraných pojmů, v němž jsou uvedeny definice vybraných pojmů.

1.1 Cíle práce

Cílem této práce je analyzovat technologie pro automatizaci běžných úkonů s prohlížeči, prozkoumat jednotlivá existující řešení pro nahrávání a přehrávání akcí, následně zjistit možnosti Puppeteeru, zejména jeho využitelnost pro nahrávání akcí a navrhnout vlastní řešení využívající k nahrávání a přehrávání akcí Puppeteer.

1.2 Struktura textu

1.3 Konvence

1.4 Využitý software a technologie

Vzhledem k tomu, že v této práci se zabýváme různými existujícími softwary a technologiemi, uvádíme zde jejich seznam, včetně konkrétně využitých verzí.

Název	Verze
Headless Recorder	0.8.1
Katalon Recorder	5.3.22
PhantomJS	2.1.1
Playwright	1.3.0
Puppeteer	5.0.0
Ranorex Recorder	9.3.4
Selenium IDE	3.17.0

Tabulka 1: Využitý software a technologie včetně verzí

2 Seznámení s technologiemi k automatizaci prohlížečů

V této části popíšeme a porovnáme vybrané technologie pro automatické ovládání prohlížečů.

2.1 Selenium

Selenium je sada nástrojů pro automatizaci prohlížečů. Jeho historie sahá až do roku 2004, kdy Jason Huggins vytvořil „JavascriptTestRunner“ pro testování webových aplikací v Pythonu. Během dalšího vývoje byl „JavascriptTestRunner“ přejmenován na Selenium (česky selen), který snižuje toxicitu rtuti. Nové jméno mělo zesměšnit tehdejší konkurenční společnost Mercury (česky rtuť). Později vznikl celý ekosystém Selenia, který zahrnuje browser drivers, selenium drivers, language bindings a testing frameworks [2, 40].

Browser driver

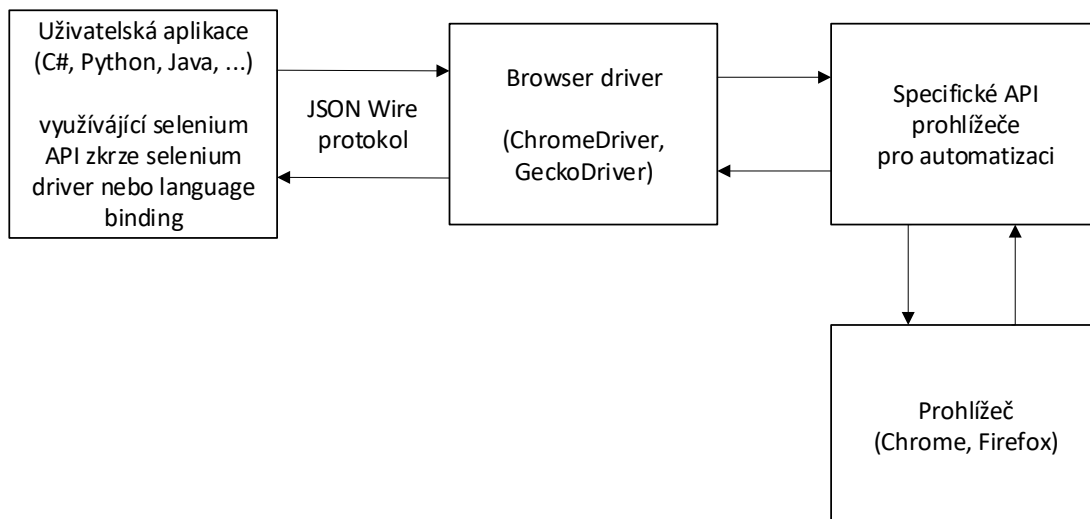
Je program vytvořený pro konkrétní prohlížeč, jehož specifické API je využíváno pro automatické ovládání. Driver dostává požadavky od uživatelské aplikace komunikující s driverem [10, 44].

Selenium driver

Jedná se o knihovnu pro určitý jazyk, která umožňuje komunikovat s browser driverem nativně pomocí selenium API přes JSON Wire protokol. [10, 44].

Language bindings

Jde se o specifický, tzv. slepovací kód, který umožňuje využívat selenium API i z jazyka, který není nativně podporován a neexistuje pro něj selenium driver [10, 44].



Obrázek 1: Komunikace Selenia s prohlížečem

Přesuneme se ke způsobu identifikace elementů na stránce. Dnes jsou v selektorech verze 3 definovány rozmanitější selektory jako např. $E \sim F$, který vybere element F , kterému předchází element E [38]. Stále však není možné napsat selektor podle vlastnosti `textContent` elementu [39]. Může se to sice zdát zprvu nedůležité, avšak často se jedná o dobrou identifikaci tlačítka, která je jednoznačná a uživatelsky přívětivá.

```
1 <button class="h51n1h-0" type="submit">Přihlásit se</button>
```

Ukázka kódu 1: HTML kod tlačítka pro přihlášení

Předpokládejme, že tlačítko obsahuje jednoznačný text uvnitř tagů, pak bychom ho mohli identifikovat textem „Přihlásit se“, namísto třídy, která může být generována na session.

Problém identifikace elementů byl v Seleniu vyřešen pomocí rozšíření selektorů na tzv. lokátory [44].

Lokátor	Identifikuje element
class name	jehož třída obsahuje hledanou hodnotu.
css selector	odpovídající hledanému selektoru.
id	jehož atribut id odpovídá hledané hodnotě.
name	jehož atribut name odpovídající hledané hodnotě.
link text	jehož viditelný text odpovídající hledané hodnotě.
partial link text	jehož viditelný text obsahuje hledanou hodnotu.
tag name	jehož název tagu odpovídá hledané hodnotě.
xpath	odpovídající XPath dotazu.

Tabulka 2: Seznam lokátorů podporovaných Seleniem [44]

Tlačítko z uk. k. 1 bychom lokátory identifikovali jako `link text=„Přihlásit se“`. Jediným však nezbytným lokátorem je XPath, s jehož pouhým využitím je možné přepsat všechny ostatní lokátory.

Mezi výhody Selenia patří podpora mnoha jazyků mezi kterými nechybí i specifitější jazyky jako např. Dart, Haskell a R. Další výhodou je dobrá podpora napříč spektrem prohlížečů, obvykle včetně podpory všech OS, na kterých daný jazyk a prohlížeč běží [10,44].

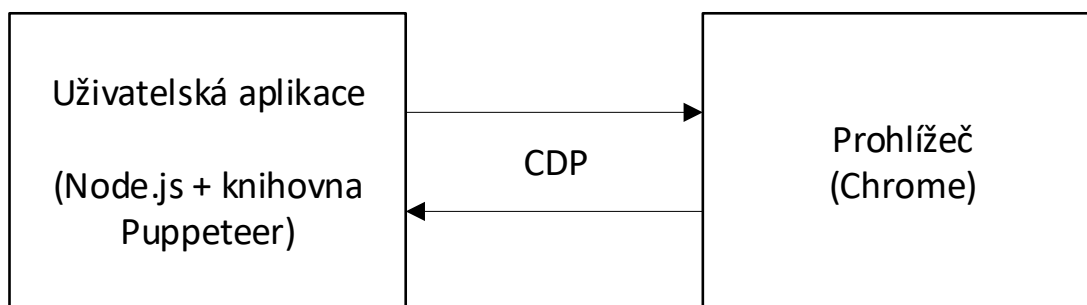
Prohlížeč	Podporované OS	Spravuje
Chromium/Chrome	Windows/macOS/Linux	Google
Firefox	Windows/macOS/Linux	Mozilla
Edge	Windows 10	Microsoft
Internet Explorer	Windows	Selenium Project
Safari	macOS El Capitan a novější	Apple
Opera	Windows/macOS/Linux	Opera

Tabulka 3: Podporované prohlížeče Seleniem [44]

Hlavní nevýhodou, částečně vyplývající z již zmíněných výhod, je omezená funkcionality selenia API, podporující pouze to, co umí všechny prohlížeče, potažmo všechny browser drivery.

2.2 Puppeteer

Puppeteer je moderní Node.js knihovna, její počátky sahají pouze do roku 2017, kdy byla vydána první verze [35]. Puppeteer poskytuje API pro ovládání Chromu¹, narozdíl od Selenia se s prohlížečem komunikuje pomocí Chrome DevTools Protocol (CDP). To hned nabízí jednu výhodu a nevýhodu. Onou výhodou je podpora specifických a proprietárních funkcí Chromu. Nevýhodou je právě však podpora pouze Chromu [31].



Obrázek 2: Komunikace Puppeteeru s Chromem

Puppeteer nelze zcela považovat za konkurenci Selenia. Selenium se zaměřuje na automatizaci nezávisle na prohlížeči. Jeho hodnotou je standardní API, které funguje ve všech hlavních prohlížečích. Puppeteer se narozdíl od toho zaměřuje na Chrome. Hodnotou Puppeteeru je bohatší funkcionalita a jednodušší API [31].

Pro každou verzi Puppeteeru existuje právě jedna verze Chromu, se kterou je garantována funkčnost.

Verze Chromu	Verze Puppeteeru
86.0.4240.0	5.3.0
85.0.4182.0	5.2.1
84.0.4147.0	5.1.0
83.0.4103.0	3.1.0

Tabulka 4: Několik odpovídajících verzí Chromu a Puppeteeru [30]

Z tohoto důvodu existují dva balíčky. Prvním je balíček „puppeteer“, který obsahuje kromě knihovny i vestavěný Chrome odpovídající verze. Druhý balíček „puppeteer-core“ obsahuje pouze knihovnu. „puppeteer-core“, která se může hodit, pokud si chceme Chrome spravovat sami nebo se Puppeteerem chceme připojit na vzdálený Chrome na jiném počítači a vestavěný by nám tak zabíral zbytečně místo [31].

¹ Aktuálně již existují nightly verze Firefoxu s FRP, jehož implementace je založená na CDP. Platí, že $FRP \subset CDP$. Je možné použít Puppeteer s Firefoxem, ačkoliv se zatím jedná pouze o experimentální podporu [36, 37].

Podíváme se na několik ukázek porovnání Node.js kódu Puppeteeru a Selenia. První ukázkou bude vytvoření screenshotu stránky seznam.cz.

```
1 const fs = require('fs')
2 const {Builder} = require('selenium-webdriver')
3 const chrome = require('selenium-webdriver/chrome');
4
5 (async () => {
6   let driver = await new Builder().forBrowser('chrome')
7     .setChromeOptions(new chrome.Options().headless())
8     .build()
9
10  await driver.get('http://seznam.cz')
11  const imageData = await driver.takeScreenshot()
12  fs.writeFileSync('C:\\scrnshot.png', imageData, 'base64')
13  await driver.quit()
14 }) ()
```

(a) Node.js Selenium

```
1 const puppeteer = require('puppeteer');
2
3 (async () => {
4   const browser = await puppeteer.launch()
5   const page = (await browser.pages())[0]
6   await page.goto('http://seznam.cz')
7   await page.screenshot({path: 'C:\\scrnshot.png', fullPage: true})
8   await browser.close()
9 }) ()
```

(b) Puppeteer

Ukázka kódu 2: Screenshot v Node.js kódu

Kód (a) i (b) spustí Chrome v headless režimu, načtou seznam.cz a uloží screenshot do souboru C:\scrnshot.png. Jediný rozdíl je ten, že kód (b) provede screenshot celé stránky, jak je nastaveno argumentem `fullPage: true`. Selenium API narozdíl od Puppeteeru neumožňuje provést screenshot celé stránky [19].

Druhá ukázka nechá vyhodnotit JavaScript uvnitř stránky prohlížeče a výsledek vypíše na výstup. Pro ušetření místa vynecháme v této a v následující ukázkách této podsekcce nedůležitý kód, zejména kód inicializující a uklízející.

```
1 const name = await driver.executeScript('return navigator.appName')
2 console.log(name)
```

(a) Node.js Selenium

```
1 const name = await page.evaluate(() => navigator.appName)
2 console.log(name)
```

(b) Puppeteer

Ukázka kódu 3: Node.js kód pro vyhodnocení JavaScriptu prohlížeče

V tomto případě si můžeme všimnout, že se kód pro Selenium a Puppeteer příliš neliší. Oba fragmenty kódu vypíší textový řetězec „Netscape“, protože hodnota `navigator.appName` je takto definována [25].

Existují však i situace, kdy je jednodušší napsat kód pro Selenium, než pro Puppeteer, viz následující ukázka.

```
1 await driver.findElement(By.linkText('About')).then(e => e.click())
2 await driver.findElement(By.name('options')).then(e => e.click())
```

(a) Node.js Selenium

```
1 await page.$x("//*[text() = 'Grading']").then(es => es[0].click())
2 await page.$x("//*[name = 'options']").then(es => es[0].click())
```

(b) Puppeteer

Ukázka kódu 4: Node.js kód pro kliknutí na element

První řádek i druhý řádek vždy obsahují kód, který nalezne a stiskne element. V případě prvního řádku se stiskne element s textovým popiskem „About“. Na druhém řádku se stiskne element obsahující atribut `name` s hodnotou „options“. Puppeteer narozdíl od Selenia nemá lokátory, elementy můžeme hledat pouze pomocí selektoru a naštěstí i XPathu [30].

2.3 Playwright

Playwright je ryze moderní záležitostí vyvíjenou společností Microsoft vydanou teprve v druhém měsíci roku 2020 [34]. Původně jde pouze o fork knihovny Puppeteer. Playwright však dává důraz na jiné aspekty funkcionality [17].

Rozdíly proti Puppeteeru

Playwright nabízí jednotné API pro automatizaci Chromu, Firefoxu a prohlížečů postavených na jádře WebKit [29]. Aktuálně však je bez úprav podporovaný pouze Chrome. Ostatní prohlížeče musí být speciálně upraveny pomocí patchů, aby bylo možné jejich použití s Playwrightem. Pro Playwright by bylo ideální sloučení těchto

patchů se zdrojovými kódy prohlížečů, v budoucnu by k tomu mohlo dojít. Některé patche již byly přijaty. [28].

Další změnou je automatické čekání na elementy, před provedením akce s nimi [29]. První ukázkou kódu pro Playwright uvedeme korektně včetně inicializace a uklízení. Všechny další ukázky této podsektce budou obsahovat pouze nezbytný kód.

```
1 const { chromium } = require('playwright');
2
3 (async () => {
4   const browser = await chromium.launch({headless: false})
5   const page = await browser.newPage()
6   await page.goto('https://webik.ms.mff.cuni.cz')
7   await page.click('#navlink_tab_grading')
8   await browser.close()
9 }) ()
```

(a) Korektní kód pro Playwright

```
1 await page.goto('https://webik.ms.mff.cuni.cz')
2 await page.click('#navlink_tab_grading')
```

(b) Nekorentní kód pro Puppeteer

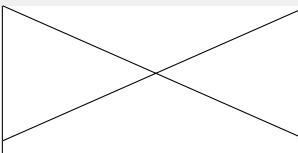
```
1 await page.goto('https://webik.ms.mff.cuni.cz')
2 await page.waitForSelector('#navlink_tab_grading')
3 await page.click('#navlink_tab_grading')
```

(c) Korektní kód pro Puppeteer

Ukázka kódu 5: Porovnání kódu Playwrightu a Puppeteeru pro stisk elementu po navigaci

Kód (a) na 7. řádku před klikem počká až element bude dostupný na stránce [27]. Naproti tomu (b) na element nečeká a pokud se nevyskytuje na stránce, tak vyhodí okamžitě výjimku [30]. Vzhledem k povaze dnešního webu je žádoucí si zkontrolovat a případně počkat na dynamické načtení elementů pomocí JavaScriptu. V Puppeteeru to můžeme zajistit viz 2. řádek (c) [30]. Povšimněme si, že speciálně kód na 1. řádku (c) nevyžaduje obdobné volání, které by počkalo na dokončení navigace. Puppeteer totiž automaticky čeká na volání „load“ události [48] uvnitř stránky [30].

Chování Playwrightu a Puppeteeru při navigaci a libovolné akci s elementem nám shrnuje následující tabulka.

Knihovna	Akce s elementem	Navigace	Vynutit čekání
Playwright	Čeká 30s, pokud se neobjeví, tak výjimka.	Čeká max 30s na zavolání „load“ události, jinak výjimka.	
Puppeteer	Pokud neexistuje, tak výjimka.		Voláním odpovídající „waitFor...“ metody.

Tabulka 5: Porovnání čekání Playwrightu a Puppeteeru [27, 30]

Jak již bylo zmíněno v 2.2 na straně 10 v Puppeteeru je možné identifikovat element pouze pomocí selektoru a XPath, což sice snižuje sílu pro identifikaci, ale nutí nás psát si vlastní nestandardní rozšíření, nebo stále dokola psát XPath dotazy. Mezi další funkce Playwrightu patří, obdobně jako lokátory v Seleniu, sofistikovanější prostředky pro identifikaci elementů [27].

```

1 await page.click('text="banán"')
2 await page.type('[data-purpose="amount"]', '42')
3 await page.check(
4   'css=form[method = "POST"] >> css=input[type = "checkbox"]')
```

Ukázka kódu 6: Akce se specificky identifikovanými elementy

Na 1. řádku se provede stisknutí elementu s textovým popiskem „banán“. Další řádek „napíše“ hodnotu 42 do elementu s atributem data-purpose = "amount". Poslední nejdelší řádek obsahuje speciální binární, zleva asociativní operátor >>. Právý operand se vyhodnotí vzhledem k elementům splňující levý operand [27]. V tomto případě se hledá checkbox pro zaškrtnutí pouze uvnitř formuláře odesílaného POST metodou. Ekvivalentně bychom mohli použít XPath:

```

1 //form[@method="POST"]//input[@type="checkbox"]
```

Ukázka kódu 7: XPath ekvivalentní identifikátoru elementu na 4. řádku

Unikátní a zajímavá funkce, kterou disponuje pouze Playwright je API pro stahování souborů. Použití je následující [27]:

1. Stránka, potažmo kontext¹ prohlížeče, ve kterém stránka běží, musí mít nastavený parametr `acceptDownloads: true`.

¹ Prostředí jedné i více stránek prohlížeče. Obvykle se sdíleným nastavením, např. neukládat historii. Při používání Chromu můžeme vytvořit nový kontext otevřením prvního okna anonymního režimu, zkratka CTRL+Shift+N. Pokud takto otevřeme další okna, nevytvoří se nový anonymní kontext, pouze se nová stránka přidá do již existujícího anonymního kontextu.

2. Při provedení akce, která způsobí stažení souboru musíme počkat na událost „download“ a zachytit zachytit objekt `Download`.
3. Nyní již libovolně můžeme využít API objektu `Download` a stažený soubor např. uložit do libovolného adresáře.

Uvedeme jednoduchý příklad, který stáhne instalátor Firefoxu a uloží ho na disk C s původním názvem.

```
1 const page = await browser.newPage({acceptDownloads: true})
2 await page.goto('https://www.mozilla.org/cs/firefox')
3
4 const [download] = await Promise.all([
5   page.waitForEvent('download'),
6   page.click('[data-download-os="Desktop"]')
7 ])
8
9 await download.saveAs('C:\\' + download.suggestedFilename())
```

Ukázka kódu 8: Stažení instalátoru Firefoxu

2.4 PhantomJS

Podíváme se ještě krátce na značně odlišnou záležitost, která již není od března roku 2018 aktivně vyvíjena [4]. Zatím jsme v kapitole 2 rozebírali jednotlivé technologie pro automatizaci existujících standardních prohlížečů. PhantomJS je však kompletní prohlížeč navržený tak, aby byl skriptovatelný. Jednou z jeho předností ve své době patřilo headless spouštění [26], které se ve Chromu objevilo až s verzí 59 pro macOS a Linux, podpora Windowsu pak byla přidána ve verzi 60 [14]. Firefox jednotně podporoval headless režim na macOS/Linuxu/Windowsu od verze 56 [45].

Prohlížeče	Datum vydání
Chrome 59	červen 2017
Chrome 60	červenec 2017
Firefox 56	listopad 2017
PhantomJS 1.0.0	leden 2011

Tabulka 6: Porovnání datumu vydání prvních verzí prohlížečů podporujících headless režim [12, 18, 32]

Z tabulky je patrné, že PhantomJS si držel konkurenční výhodu dlouhých 6 let. Po její ztrátě již netrvalo dlouho a projekt byl archivován.

Na závěr si ukážeme fragment kódu využívající Node.js binding pro PhantomJS. Provedeme screenshot celé stránky `mff.cuni.cz` a uložíme ho jako `C:\scrnshot.pdf`.

```
1 const phantom = require('phantom');
2
3 (async () => {
4   const browser = await phantom.create()
5   const page = await browser.createPage()
6   await page.open('https://mff.cuni.cz')
7   await page.render('C:\\scrnshot.pdf')
8   await browser.exit()
9 }) ()
```

Ukázka kódu 9: Uložení screenshotu celé stránky do PDF s využitím PhantomJS

Obdobně jednoduchým způsobem je možné vytvořit screenshot s uložením do PDF i s použitím Puppeteeru nebo Playwrightu. V případě použití Selenia je nutné se omezit pouze výstup v obrázkovém formátu [44] a na screenshot viewportu, jak již bylo zmíněno v 2.2 Puppeteer na straně 9.

3 Nahrávání a přehrávání akcí

V této kapitole se podíváme na řešení sloužící k zaznamenávání uživatelských akcí v prohlížeči s možností následného opětovného vykonání, dále pouze uváděno jako řešení pro nahrávání a přehrávání akcí nebo pouze řešení.

Nejprve se podíváme na vybraná existující konkurenční řešení, později se zaměříme výhradně na Puppeteer, pro který popíšeme jeho možnosti pro nahrávání a přehrávání akcí včetně způsobu využití.

3.1 Konkurenční řešení

V této části se výhradně zaměříme na konkurenční řešení. Podíváme se na jejich funkcionalitu, popíšeme výhody a nevýhody.

Přikládáme ještě tabulku primárně určenou pro rychlou orientaci v této sekci a letmé porovnání.

Název	Využívá	Licence	Nahrávání	Přehrávání
Selenium IDE	Selenium	Apache License 2.0	✓	✓
Katalon Recorder	Selenium	Apache License 2.0	✓	✓
Ranorex Recorder	Proprietární		✓	✓
Headless Recorder	Puppeteer	Apache License 2.0	✓	✗

Tabulka 7: Orientační porovnání existujících řešení

3.1.1 Selenium IDE

Selenium IDE je nástroj postavený nad Seleniem viz 2.1 fungující jako rozšíření Chromu a Firefoxu. Původně ho navrhl Shinya Kasatani, který umožnil v roce 2006 jeho vstup mezi oficiální nástroje Selenia [15, 41].

V roce 2017, kdy byla aktuální v2 Selenia IDE, se projekt dostal do problémů, kód v té době fungoval pouze s Firefoxem a byl závislý na specifickém API Firefoxu pro add-onsy [1]. Těmto add-onům se dnes říká „legacy extensions“, ve Firefoxu byla jejich podpora úplně odstraněna s verzí 57 [13], která vyšla ve stejném roce [12]. Projekt se však díky práci vývojářů podařilo udržet naživu.

Aktuálně existují dvě vyvíjené větve, první je v3, ve které se stále ještě přepisují některé části kódu starého Selenia IDE v2. Cílem v3 je podpora moderních aktuálních verzí prohlížečů Chrome a Firefox včetně všech funkcí původního Selenia IDE v2 [15].

Druhou větví je master, ve které se pracuje na přechodu z rozšíření Chromu a Firefoxu na Electron. V současné době však zatím nebyla vydána žádná verze fungující s Electronem. Zájemci o vyzkoušení mají možnost si zdrojový kód

zkompilevat sami [15].

Léta vývoje se projevují na funkcionalitě, pojďme se na některé zajímavé funkce Selenia IDE v3.17.0, současně nejnovější verze dostupné [33], podívat.

Výběr z lokátorů

Pro každý element se nalezne sada lokátorů, uživatel si může vybrat ten nejvhodnější.

	Command	Target
1	open	/
2	set window size	1036x1004
3	click	id=navlink_tab_labs

Command

click

//

Target

id=navlink_tab_labs

Value

id=navlink_tab_labs

id

Description

linkText=Labs

linkText

css=#navlink_tab_labs

css:finder

xpath=//a[contains(text(), 'Labs')]

xpath:link

xpath=//a[@id='navlink_tab_labs']

xpath:attributes

xpath=//div[@id='header']/nav/a[4]

xpath:idRelative

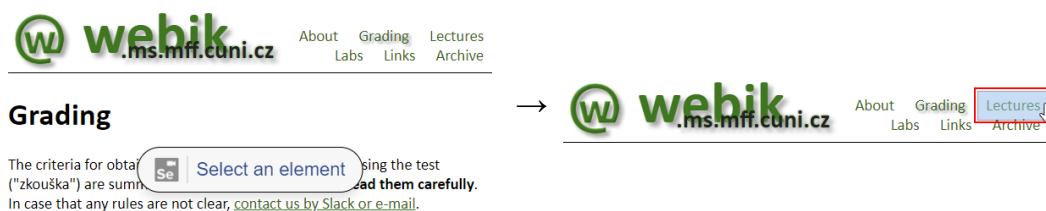
xpath=//a[contains(@href, 'Labs')]

xpath:href

Obrázek 3: Výběr z lokátorů identifikujících element

Změna elementu klikem myši

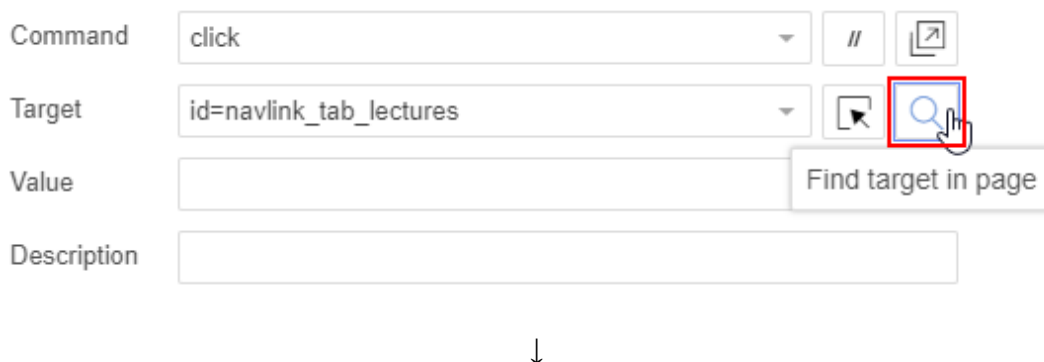
Pro změnu elementu na němž se má akce provést, aniž bychom museli ručně přepisovat lokátor, existuje tlačítko umožňující znovu zachytit element.



Obrázek 4: Změna elementu bez ručního přepisování lokátoru

Zobrazení nalezeného elementu

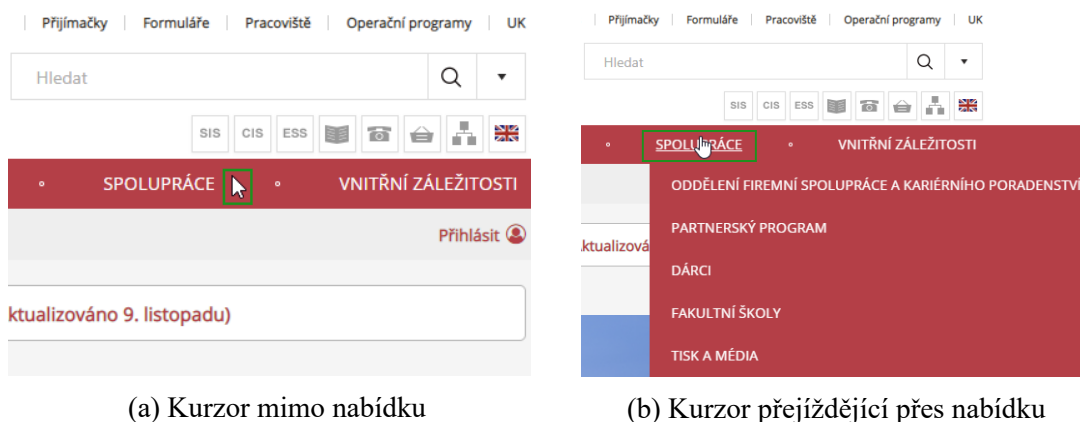
Pokud chceme zpětně zjistit jaký element je lokátorem identifikován, stačí kliknout na tlačítko s obrázkem lupy, identifikovaný element se uvnitř stránky zvýrazní.



Obrázek 5: Zpětné zobrazení identifikovaného elementu

Mouseover

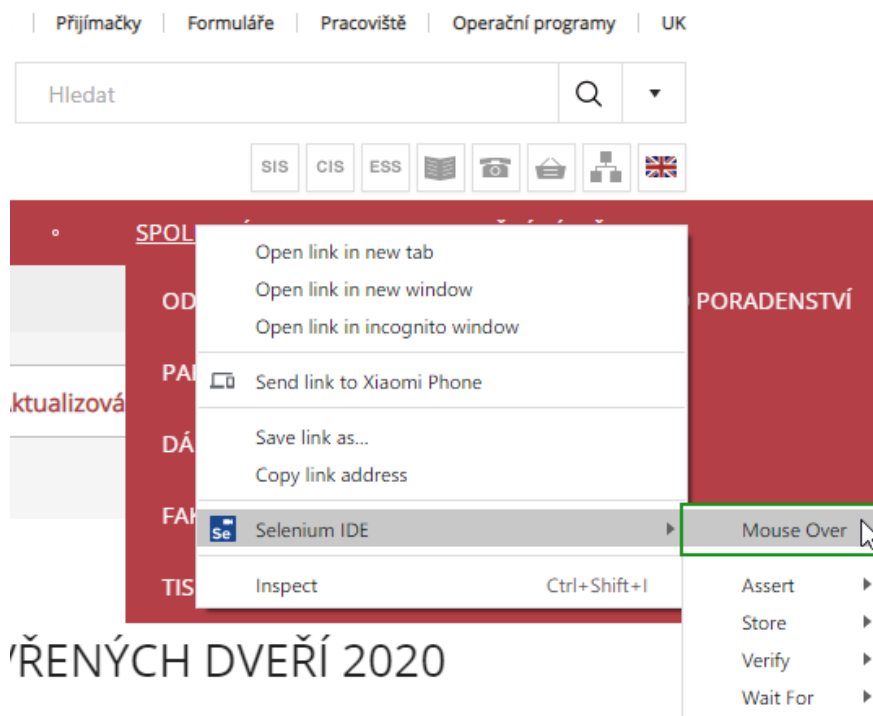
Mnoho elementů na stránce je často skryto, dokud nepřejedeme myší přes nabídkový element, jehož JavaScript způsobí odkrytí elementů.



Obrázek 6: Stránky MFF UK s nabídkovým elementem

Není vhodné zaznamenávat veškeré pohyby kurzoru přes elementy. Většina z nich nezpůsobuje žádný efekt na stránce, pouze nám zahlcují nahrávač přebytečnými událostmi a snižují přehlednost. Jak však nahrát nezbytná přejetí kurzoru přes element? Ukážeme si jak je tento problém vyřešen v Selenium IDE.

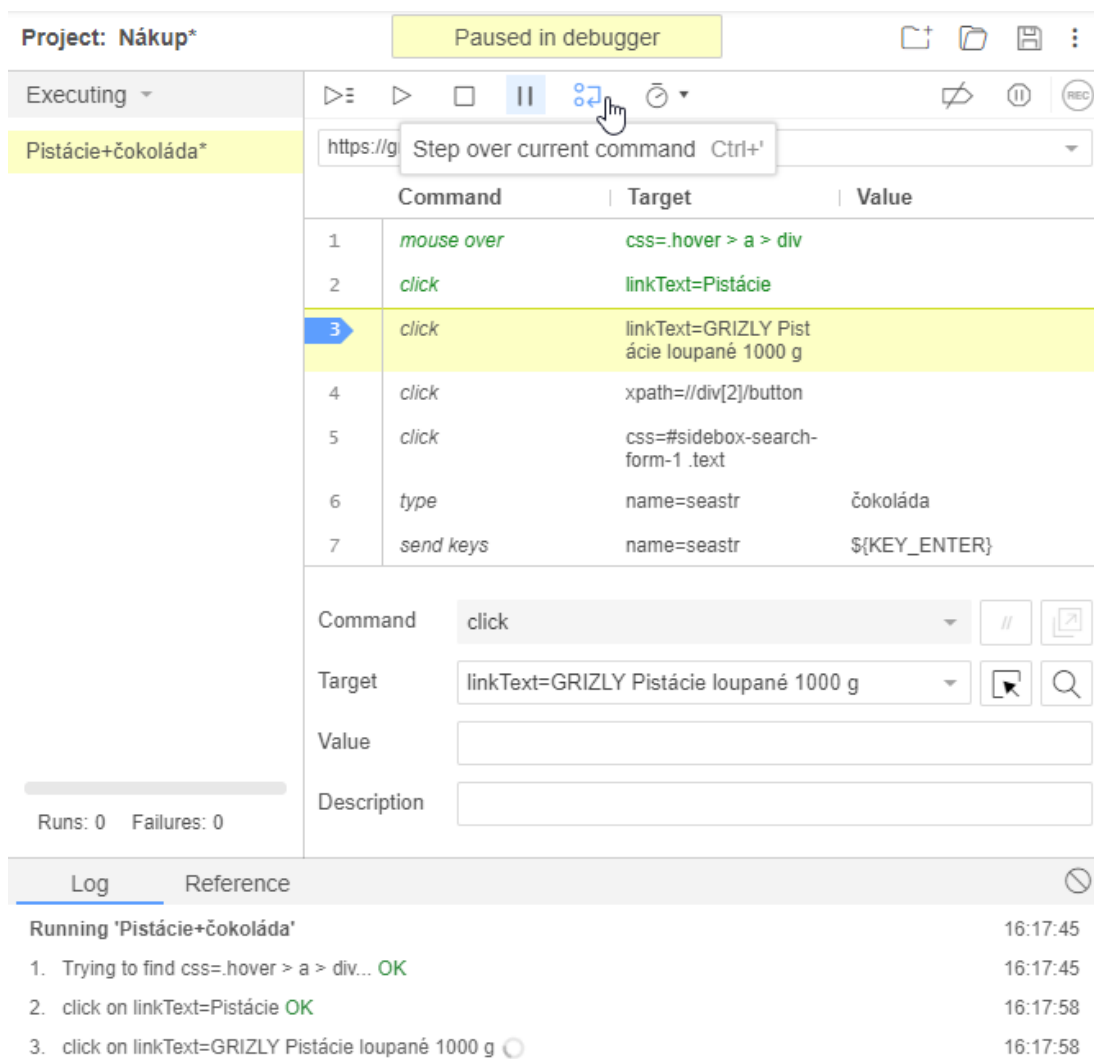
Implicitně se nezaznamenávají žádné „mouseover“ události. Zaznamenání si však můžeme explicitně vyžádat.



Obrázek 7: Explicitní zaznamenání „mouseover“ akce

Debugger

Selenium IDE se tak nejmenuje náhodně. Jeho součástí je plnohodnotný debugger, včetně breakpointů, krokování, tak jak ho můžeme znát z IDE.



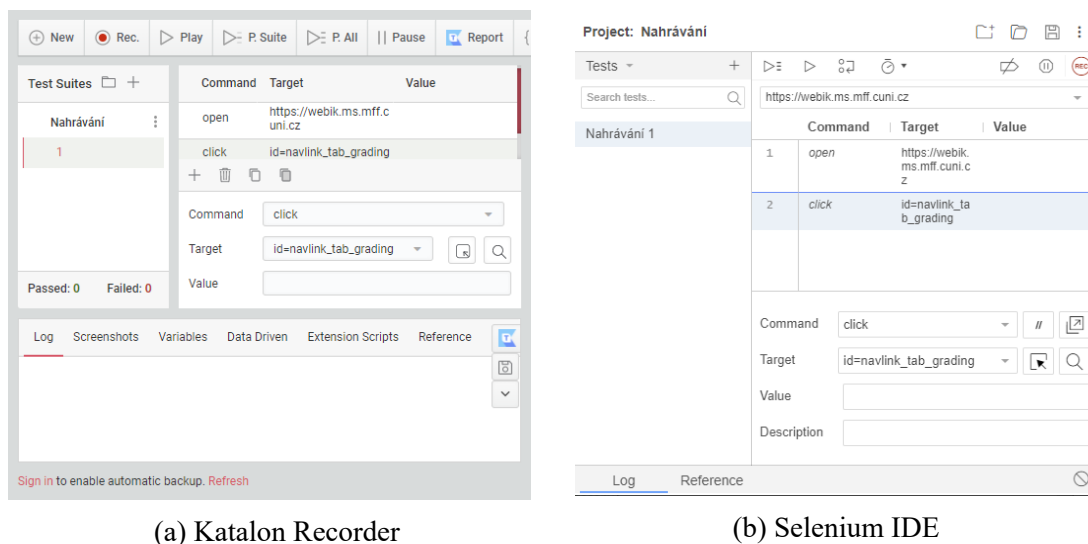
Obrázek 8: Přehrávání s breakpointem na 3. akci

Obrátme pohled na nevýhody, mezi větší nevýhodu aktuálního Selenia IDE v3, než by se na první pohled mohlo zdát, patří jeho návrh závislý na API prohlížečů pro extensiony. Dnes v3 je extension pro Chrome i Firefox, oproti v2, která byla závislá na specifickém API Firefoxu, fungující pouze s Firefoxem. Nicméně úplné osvobození od API prohlížečů přijde až s vydáním verze postavené na Electronu.

Kritickou záležitostí aktuálního Selenia IDE v3 je, že může komunikovat pouze s instancí prohlížeče, která ho spustila, tzn. že nemůže komunikovat s prohlížečem bez nainstalovaného extensionu, případně s prohlížečem běžícím na vzdáleném počítači.

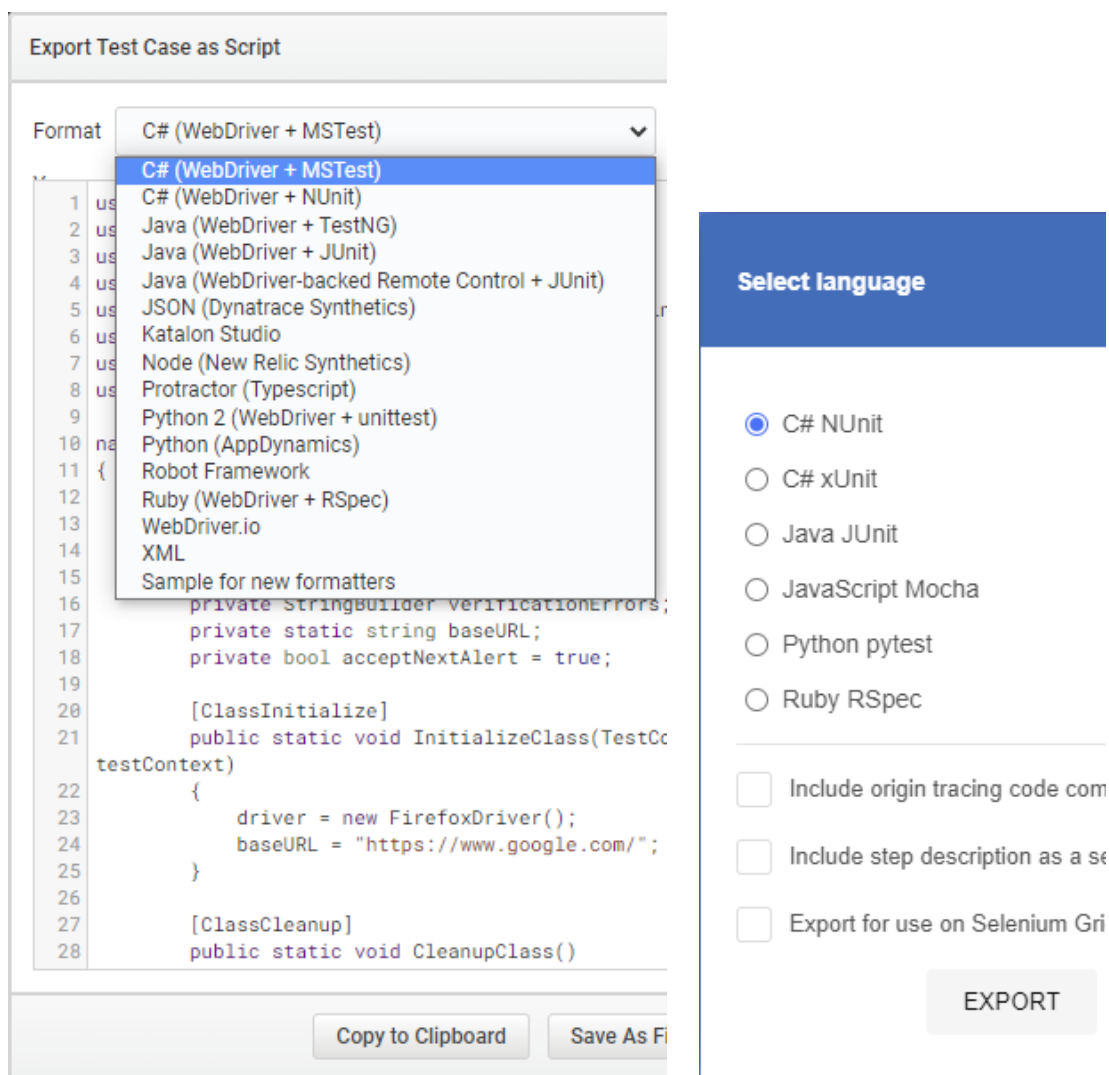
3.1.2 Katalon Recorder

Katalon Recorder je produkt společnosti Katalon LLC. Původně jde o fork Selenia IDE, již od vydání první veřejné verze v roce 2016 Katalon Recorder fungoval s Chromem a Firefoxem [22, 24]. Zejména v roce 2017, kdy Selenium IDE fungovalo pouze s Firefoxem a využívalo jeho historické, odstraněné API, byl Katalon Recorder velmi silný významný konkurent. Vzhledem k tomu, že se dnes Selenium IDE ze svých problémů dostalo, se jedná o dvě velmi podobná a srovnatelná řešení.



Obrázek 9: Porovnání GUI Katalon Recorderu a Selenia IDE

Hlavním důvodem pro upřednostnění Katalon Recorderu při výběru mezi různými existujícími řešeními je jeho placená varianta Katalon Studio Enterprise, ve které je zahrnuta podpora produktu, ale i rozšířené funkce. Jednou takovou funkcí je „self-healing execution“, tj. využití nalezených náhradních lokátorů, pokud vybraný selže [23]. Existují ale i další méně patrné funkce Katalon Recorderu zahrnuté i ve volně dostupné verzi, které Selenium IDE nemá.



(a) Katalon Recorder

(b) Selenium IDE

Obrázek 10: Porovnání podporovaných jazyků pro export

Jak obr. 10 výše ukazuje, Katalon Recorder dokáže exportovat nahrané akce do většího počtu různých technologií, narozdíl od Selenium IDE.

Na druhou stranu v Katalon Recorderu jsou dostupné pouze tři lokátory a těmi jsou id, name a xpath [21] (seznam lokátorů viz tab. 2 na straně 7). Ostatní jsou zpřístupněny až v Katalon Studiu Enterprise a Katalon Studiu [21], což je bezplatná varianta Katalon Studia Enterprise postrádající mimo jiné podporu. Ani tuto variantu není možné stáhnout bez vytvoření Katalon účtu vyžadujícího jméno, email, heslo a zodpovězení dvou otázek [23].

3.1.3 Ranorex Recorder

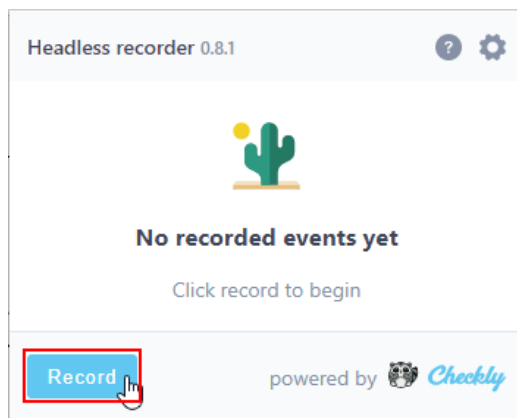
Velmi krátce zmíníme jedno proprietární komerční řešení, tím je Ranorex Recorder, který je součástí balíčku Ranorex Studio sloužící pro automatizaci GUI a prohlížečů. Původně byl tento produkt vytvořen v Rakousku v roce 2007 jakožto volně dostupná

aplikace. Následná popularita aplikace vedla k vývoji komerční verze s podporou, která byla v roce 2017 odkoupena americkou společností IDERA, Inc [49].

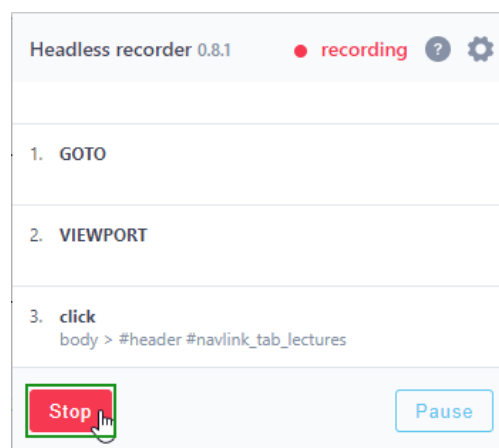
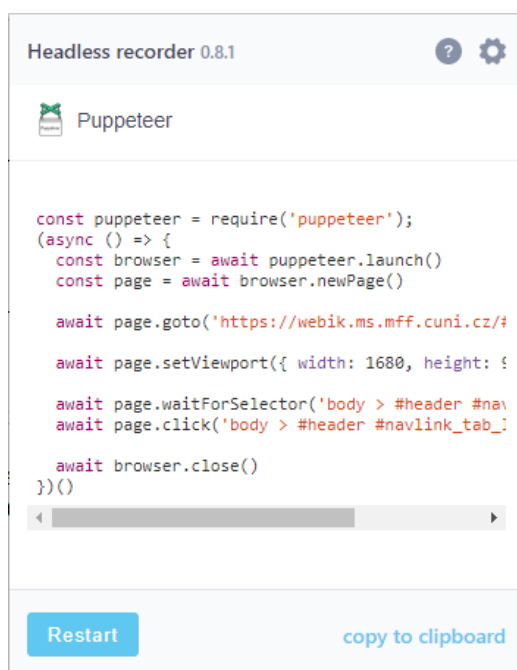
Pomineme GUI a zaměříme se pouze na automatizaci prohlížečů, její funkčnost je zajištěna na bázi komunikace externí aplikace Ranorex Recorderu a prohlížeče resp. jeho rozšíření Ranorex Automation. Mezi prohlížeče, pro které existuje rozšíření Ranorex Automation a jsou proto podporovány, patří Internet Explorer, Firefox a Chrome.

3.1.4 Headless Recorder (dříve Puppeteer Recorder)

Jediným řešením postaveným na Puppeteeru, o kterém se zmíníme je Headless Recorder od společnosti Checkly, Inc. Řešení je postavené nad Puppeteerem, a proto je kompatibilní pouze s Chromem. Celá implementace je provedena jako rozšíření prohlížeče, nejedná se o úplné řešení umožňující nahrávání a přehrávání akcí. Podporované je pouze zaznamenávání několika málo akcí a vygenerování Puppeteer kódu [16].



→ Zaznamenáme akce, jejichž kód chceme vygenerovat.



Obrázek 11: Použití Headless Recorderu

Funkcionalita

Mezi zaznamenávané akce patří základní události jako stisk klávesy a myši. Podpora „mouseover“ události v jakékoliv podobě chybí stejně jako podpora lokátorů či případných jiných rozšíření selektorů.

V nastavení je k dispozici, jak ukazuje následující obrázek, pouze několik položek, jejichž význam je zřejmý.

Headless Recorder Options

Code Recorder settings

CUSTOM DATA ATTRIBUTE

your custom data-* attribute

Define an attribute that we'll attempt to use when selecting the elements, i.e "data-custom". This is handy when React or Vue based apps generate random class names.


Code Generator settings

- ☒ Wrap code in async function
- ☒ Set headless in puppeteer launch options
- ☒ Add `waitForNavigation` lines on navigation
- ☒ Add `waitForSelector` lines before every `page.click()`
- ☒ Add blank lines between code blocks

Extension settings

- ☐ Allow recording of usage telemetry

We only record clicks for basic product development, no website content or input data. Data is never, ever shared with 3rd parties.

sponsored by  Checkly

Obrázek 12: Nastavení Headless Recorderu

Právě jedním ze zásadních důvodů pro vytvoření této práce byly chybějící pokročilé funkce jako např. nemožnost přehrát zaznamenané akce.

3.2 Využití Puppeteeru

Uvědomme si, že Puppeteer je určen pro automatizaci prohlížeče. Naším cílem je využít ho pro nahrávání a přehrávání akcí s možností zobrazení vygenerovaného kódu pro Puppeteer. Z důvodů tohoto neobvyklého použití se objevilo několik problémů, které ukážeme spolu s popisem využití Puppeteeru pro tento účel.

Podíváme se na jednotlivé akce s Chromem, rozdělíme si akce na dvě skupiny. První bude zahrnovat akce s oknem prohlížeče, např. otevření tabu, zavření tabu, přepnutí aktivního tabu apod. Tyto akce nazveme „WindowEvents“. Druhou skupinou zvanou „ViewportEvents“ budeme rozumět akce odehrávající uvnitř viewportu prohlížeče, např. kliknutí na element, rolování stránky, odeslání formuláře apod.

V dalších částech budeme demonstrovat způsoby zachycení a kód k vygenerování pro opětovné přehrávání akcí. Ukázky kódu budou postrádat nezajímavé rušivé části jako inicializace apod. Pokud by byl kód příliš dlouhý nebo ho nepovažujeme za nezbytný, budeme odkazovat do appendixu A, kde jsou všechny takové ukázky umístěny.

3.2.1 WindowEvents

Začneme s přímočarými událostmi, pro které je v Puppeteeru připraveno dobře fungující API. Poté se přesuneme k akcím, které se nedají zachytit triviálním způsobem nebo mají složitý kód pro opětovné přehrávání.

Otevření nového tabu

```
1 browser.on('targetcreated', target => {
2   if (target.type() === 'page')
3     console.log('Vytvořen nový tab s~url: ' + target.url())
4 })
```

Ukázka kódu 10: Zachycení otevření nového tabu

Vzhledem k tomu, že typ proměnné `target` nemusí být vždy „page“, jež odpovídá otevření nového tabu, je podmínka na 2. řádku potřebná.

Obdobně jednoduchý je i kód pro přehrávání. Malým nedostatkem je, že API Puppeteeru neumožňuje otevřít nový tab s konkrétní adresou URL, proto musíme hned po otevření adresu změnit.

```
1 const newPage = await browser.newPage()
2 await newPage.goto(url)
```

Ukázka kódu 11: Otevření nového tabu

Uzavření existujícího tabu

```
1 browser.on('targetdestroyed', target => {
2   if (target.type() === 'page')
3     console.log('Uzavřen existující tab s~url: ' + target.url())
4 })
```

Ukázka kódu 12: Zachycení uzavření existujícího tabu

```
1 await page.close()
```

Ukázka kódu 13: Uzavření existujícího tabu

Kód uk. k. 13 výše uzavře objekt `page`, kterému odpovídá nějaký tab. Abychom věděli, který tab máme uzavřít, potřebujeme jeho jednoznačnou identifikaci. API Puppeteeru žádnou takovou identifikaci nedisponuje. Jedinou naší možností je vytvoření vlastního unikátního identifikátoru. Abychom měli jistotu, že každý tab obsahuje identifikátor, nejlepší je přidělit mu ho hned po vytvoření tabu, např. v události `targetcreated`.

Změna adresy URL

```
1 browser.on('targetchanged', async target => {
2   if(target.type() === 'page') {
3     const oldUrl = (await target.page()).url()
4     const newUrl = target.url()
5     console.log('Změna URL z~' + oldUrl + ' na ' + newUrl)
6   }
7 })
```

Ukázka kódu 14: Zachycení změny adresy URL tabu

Provedení změny adresy URL jsme už viděli v uk. k. 10. Pro připomenutí: stačí zavolat členskou metodu `goto` objektu `page` a jako jediný parametr předat adresu URL.

Změna aktivního tabu

Pro zachycení změny aktivního tabu bohužel neexistuje žádné API v Puppeteeru. Dokonce není ani možné zjistit, který tab je momentálně aktivní. Puppeteer nám však umožňuje vykonat vlastní JavaScript uvnitř stránky s vrácením výsledku. Toho můžeme využít a na všech otevřených stránkách zavolat

`document.visibilityState === 'visible'`, aktivní stránka by nám měla vrátit `true`. Problémem však zůstává, jak se o změně aktivního tabu dozvědět v momentě, kdy se změnil aktivní tab. To s pouhým využitím Puppeteeru nejde a musíme se spokojit s následujícím:

- Po otevření nového tabu je nově otevřený tab aktivním tabem.
- Při zachycení události ohlašující změnu adresy URL je nezbytné ověřit, zda se nezměnil aktivní tab.
- Při zachycení události ze skupiny „ViewportEvents“ je nutné zkontrolovat zda před ní nedošlo ke změně aktivního tabu.

Dodržením předchozích bodů se spolu se zachycenou událostí dozvíme o již provedené změně aktivního tabu.

A.2 obsahuje kód, který nám vrátí momentálně aktivní tab, myšlenka kódu byla inspirována diskuzí na GitHubu [50].

Situace je zcela odlišná, pokud chceme nastavit určitý tab jako aktivní. Pro tento úkon existuje v API Puppeteeru metoda.

```
1 await page.bringToFront()
```

Ukázka kódu 15: Nastavení tabu jako aktivního

3.2.2 ViewportEvents

Jak již víme, API Puppeteeru nám umožňuje vykonat JavaScript uvnitř Chromu. Navíc, a to je pro zachycení událostí z viewportu nepostradatelné, můžeme „zveřejnit“ libovolnou metodu našeho kódu do `window` objektu Chromu. To znamená, že uvnitř okna prohlížeče můžeme nechat vykonat kód, který zavolá zpátky metodu (i včetně předání parametrů) v našem kódu pracujícím s Puppeteerem.

Případné zájemce o kód odkážeme do appendixu – A.1 obsahuje společnou kostru pro zachycení všech akcí. Součástí A.1 je i popis částí kostry. Upozorňujeme však, že kód postrádá zejména algoritmy pro získání identifikátorů elementů jakými jsou selektory nebo lokátory.

Přesuneme se ke způsobům přehrávání získaných akcí, jak již množné číslo naznačuje, způsoby se budou lišit, a to nejen podle druhu akce. API Puppeteeru umí pracovat pouze se selektory, pokud chceme pracovat s lokátory, musíme si vytvořit vlastní metody, které je převádějí na XPath. Nemožnost využití jednotného API nezávisle na způsobu identifikace elementu vyžaduje neustále dodávat vlastní rozšíření ke standardní verzi Puppeteeru, nebo využívat nestandardní vlastní verzi Puppeteeru opatřenou již o tato rozšíření. Ani jedna z těchto variant není ideální, s využitím první

varianty alespoň nepřijdeme o kompatibilitu, z tohoto důvodu se domníváme, že první varianta je lepší volbou.

Nyní ukážeme konkrétní způsoby přehrávání akcí, pro jednoduchost se omezíme pouze na identifikaci pomocí selektorů. Metody převádějící vybrané lokátory na XPath s návratovou hodnotu odpovídajících elementů jsou k nahlédnutí v A.4.

Kliknutí

API Puppeteeru obsahuje metodu pro provedení kliknutí.

```
1 await page.click(selector)
```

Ukázka kódu 16: Kliknutí na element odpovídající selektoru

Přejetí myši

Situace je zde obdobná jako pro kliknutí.

```
1 await page.hover(selector)
```

Ukázka kódu 17: Přejetí myši na element odpovídající selektoru

Odeslání formuláře

Pro odeslání formuláře neexistuje API Puppeteeru. Víme, že je možné nechat vykonat JavaScript uvnitř prohlížeče, což se nám hodí, protože Web API disponuje jednoduchou metodou `submit` pro odeslání formuláře [51].

```
1 await page.$eval(selector, form => form.submit())
```

Ukázka kódu 18: Odeslání formuláře pomocí Web API

Dle API Puppeteeru, v uk. k. 18 odpovídá proměnná `form` návratové hodnotě `document.querySelector(selector)`.

Rolování

Nápodobně jako u odesílání formuláře i rolování je možné provést pomocí Web API. `window` objekt pro rolování stránky zahrnuje hned několik funkcí k tomuto účelu: `scrollTo`, `scrollBy`, `scrollByLines` a `scrollByPages` [52]. První funkce z tohoto seznamu roluje stránku absolutně vůči levému hornímu rohu, ostatní fungují relativně vůči aktuálnímu stavu.


```
1 await page.evaluate(() => window.scrollTo(x, y))
```

Ukázka kódu 19: Rolování stránky

Pokud bychom potřebovali rolovat nikoliv celou stránku ale element, máme k dispozici funkce `scrollTo` a `scrollBy` [53], které fungují stejně jako varianty pro `window` objekt.

Dvojitě kliknutí

Zde opět nemáme podporu API Puppeteeru a tak musíme využít Web API. Připomeňme si nejprve obyčejné kliknutí, to je možné provést voláním členské funkce `click` objektu `HTMLElement` [54]. Takto se kliknutí provede, i když využije jeho podporu v API Puppeteeru.

Pro dvojitě kliknutí bychom očekávali existenci metody `dblclick`, ať už ve Web API nebo v API Puppeteeru. Taková metoda neexistuje a proto musíme:

1. Vytvořit událost reprezentující dvojitě kliknutí.
2. Získat element, na který událost aplikujeme.
3. Spustit vytvořenou událost se získaným elementem.

```
1 await page.evaluate(() => {  
2   let evt = new Event('dblclick')  
3   let el = document.querySelector(selector)  
4   el.dispatchEvent(evt)  
5 })
```

Ukázka kódu 20: Dvojitě kliknutí na element odpovídající selektoru

Označení textu

Označením textu rozumíme událost „select“, která je dostupná pouze pro elementy `<input type="text">` a `<textarea>`. Touto událostí se zaobíráme, protože chceme získat možnost zaznamenat a opětovně simulovat výběr textu, který se může měnit. Navíc po výběru textu často následuje událost kopírování (zápis textu do schránky). Kopírovat pak můžeme aktuální výběr namísto staticky uloženého textu.

```

1 await page0.$eval(selector, el => {
2   el.setSelectionRange(event.selectionStart,
3                         event.selectionEnd,
4                         event.selectionDirection)
5 })

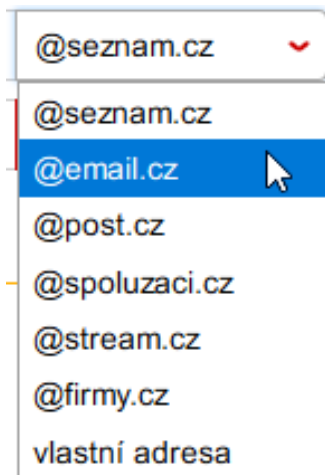
```

Ukázka kódu 21: Výběr textu

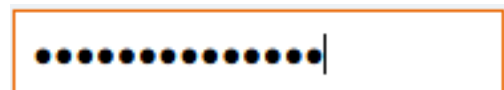
V uk. k. 21 předpokládáme, že proměnná `event` obsahuje informace o události „select“. Konkrétně `selectionStart` i `selectionEnd` jsou indexy textu. V `selectionStart` výběr textu započal, znak na indexu `selectionEnd` již do výběru nepatří. `selectionDirection` udává směr, kterým byl výběr proveden.

Událost „change“

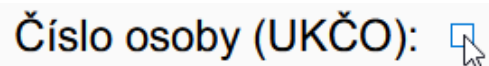
Problémem této události je, že zahrnuje změny hodnot v různých typech elementů.



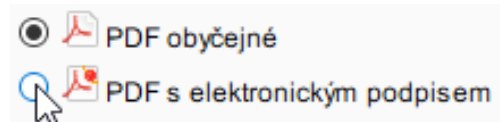
(a) `<select>`



(b) `<input type="password">`



(c) `<input type="checkbox">`



(d) `<input type="radio">`

Obrázek 13: Několik elementů, jejichž změna hodnoty vyvolá událost „change“

Puppeteer nám dává k dispozici metodu `select`, která umí vybrat hodnotu v případě (a). Pro přehrání vytváření hesla v (b) existuje metoda `type` které můžeme nastavit i volitelné zpoždění mezi jednotlivými údery.

```

1 page.select(selector, ...values)

```

(a)

```

1 page.type(selector, text, {delay: durationMs})

```

(b)

Ukázka kódu 22: Přehrávání události „change“

Pro poslední dva elementy obr. 13(c) a 13(d) je možné použít kliknutím, které je možné provést nám již známou metodou `click`.

Zápis/čtení textu do/ze schránky (události „copy“ a „paste“)

Předtím, než se začneme zabývat kopírováním a vkládáním, bychom rádi uvedli čtenáře do aktuálního stavu udělování oprávnění pomocí CDP, ať už nepřímo použitím Puppeteeru nebo nativně s pouhým využitím CDP.

Dle Puppeteer API by mělo být možné udělit oprávnění „push“, nicméně při pokusu o získání tohoto oprávnění je program ukončen s výjimkou „Error: Unknown permission: push“. Pro ověření tohoto tvrzení je možné spustit kód v A.3.

S oprávněními Puppeteeru „clipboard-read“ a „clipboard-write“ pro práci se schránkou je situace nápodobná jako u oprávnění „push“. Zřejmě bychom očekávali, že bez oprávnění „clipboard-read“ („clipboard-write“) není možné číst (měnit) obsah schránky. Tak tomu ale není.

Experimentálním ověřením s nahlédnutím do zdrojového kódu Puppeteeru a online dokumentace CDP [9] jsme zjistili, že:

- Při neudělení žádného oprávnění je možné zapisovat do schránky, při pokusu o čtení se zobrazí výzva pro udělení/neudělení oprávnění pro čtení. Připravený kód, kterým je možné toto ověřit se nachází v A.5.
 - Puppeteer nabízí oprávnění „clipboard-read“ a „clipboard-write“, CDP definuje oprávnění „clipboardReadWrite“ a „clipboardSanitizedWrite“. Při pokusu o získání oprávnění „clipboard-read“ nebo „clipboard-write“ Puppeteer vždy zavolá CDP funkci `Browser.grantPermissions` s parametrem `permissions` obsahující pouze „clipboardReadWrite“.
- Volání této funkce, ať už přímo nativně nebo nepřímo s použitím Puppeteeru, má za efekt to, že při pokusu zápisu do schránky se program ukončí s výjimkou „Error: Evaluation failed: DOMException: Write permission denied.“, více viz A.6. Čtení naopak funguje bezproblémů viz A.7.
- API Puppeteeru potažmo Puppeteer nijak nevyužívá oprávnění „clipboardSanitizedWrite“ definované CDP, avšak udělením tohoto oprávnění získáme práva k zápisu. Konečná funkční ukázka se nachází v A.8.

Následující dvě tabulky shrnují chování.

Oprávnění CDP	Odpovídající oprávnění Puppeteeru
clipboardReadWrite	clipboard-read clipboard-write
clipboardSanitizedWrite	

Tabulka 8: Oprávnění pro schránku definované CDP a Puppeteerem

Udělená oprávnění	Čtení	Zápis
bez oprávnění	✗	✓
clipboardReadWrite	✓	✗
clipboardSanitizedWrite	✗	✓
clipboardReadWrite a clipboardSanitizedWrite	✓	✓

Tabulka 9: Skutečný význam oprávnění CDP pro schránku

Důvodem těchto problémů může být, že online dokumentace CDP [9] uvádí celé API pro oprávnění jako experimentální.

4 Řešení

V této kapitole se detailně zaměříme na vlastní řešení navržené s využitím Puppeteeru. Nejprve si popíšeme z jakých částí ze řešení skládá a jak tyto části mezi sebou komunikují, včetně diskuze o vybraných technologiích. Poté se podíváme na funkčnost a porovnáme ji vůči konkurenčním produktům. Dále v této kapitole – konkrétně v části 4.4 se nachází dokumentace pro uživatele. Pro zájemce o detailnější popis zdrojového kódu je připravena podsektce 4.5.

4.1 Design

Řešení se skládá ze dvou celků. Prvním je knihovna pro přehrávání a nahrávání akcí vytvořená s běhovým prostředím Node.js v JavaScriptu, dále označovaná jako backend. Vzhledem k implementaci Puppeteeru v TypeScriptu¹ jsme se rozhodli pro JavaScript. Druhým celkem je okenní WinForms aplikace implementovaná v C# pro poskytnutí UI, tu nazveme frontend. C# a WinForms jsme vybrali z těchto důvodů:

1. Jednoduchost

Jak uvádí dokumentace firmy Microsoft [56], která stojí za vznikem C#, jedním z designových cílů již od verze 1.0 byla snaha o „jednoduchý, moderní, univerzální objektově orientovaný jazyk“. Stejně jako C# i návrh UI ve WinForms považujeme za jednoduchý, a to zejména díky obrovskému množství již existujících komponent (ProgressBar, PropertyGrid, ...) a vývoji pomocí drag and drop. Cílem této práce není vytvářet UI podle grafického návrhu a navíc WinForms obsahuje většinu námi potřebných komponent a všechny nutné funkce.

2. Přenositelnost

Jazyk C# byl vydán s vývojovým prostředím Visual Studio .NET 2002 [56], současná verze Visual Studio 2019 je stále kompatibilní pouze s Windows [58], zatímco programy vytvořené v C# s WinForms je možné spouštět i na jiných platformách (Linux, macOS, BSD a další), díky projektu Mono² [57].

3. Vyzrálост

Součástí první verze .NET Framework vydané v únoru 2002 byl WinForms. Dnes je stále její součástí, ač je tomu nyní již 19 let. O výběru technologie pro UI by

¹ JavaScript se statickým typováním spravovaný firmou Microsoft [55].

² Multiplatformní implementace .NET Framework s otevřeným zdrojovým kódem [57].

se jistě dalo diskutovat – pouze samotný Microsoft stihl vydat WPF a UWP, WPF slouží výhradně pro vývoj UI [64] a UWP je, spolu s vývojem UI, přímo univerzální platforma pro vývoj Windows aplikací [65]. Narozdíl od WinForms, WPF i UWP fungují pouze na Windows¹.

Na Internetu se často vyskytují otázky týkající se končícího životního cyklu WinForms [59–63]. Místo souhlasu s tím, že WinForms jsou mrtvé, se domníváme, že jejich stárí přispívá ke stabilitě a neměnosti API, které může i způsobit ztrátu kompatibility mezi jednotlivými verzemi.

Backend a frontend mezi sebou potřebují vzájemně komunikovat. Např. frontend spouští nahrávání, tj. odesílá pokyn ke spuštění nahrávání backendu. Naopak backend při nahrávání ohlašuje zachycenou událost do frontendu. Pro komunikaci mezi dvěma procesy bychom mohli využít BSD sockety, které mají nativní podporu v API Node.js [68] i v API .NET Frameworku [69], avšak pro tyto účely již existuje řada knihoven.

Vybrali jsme knihovnu ZeroMQ [70] pro tyto její výhody:

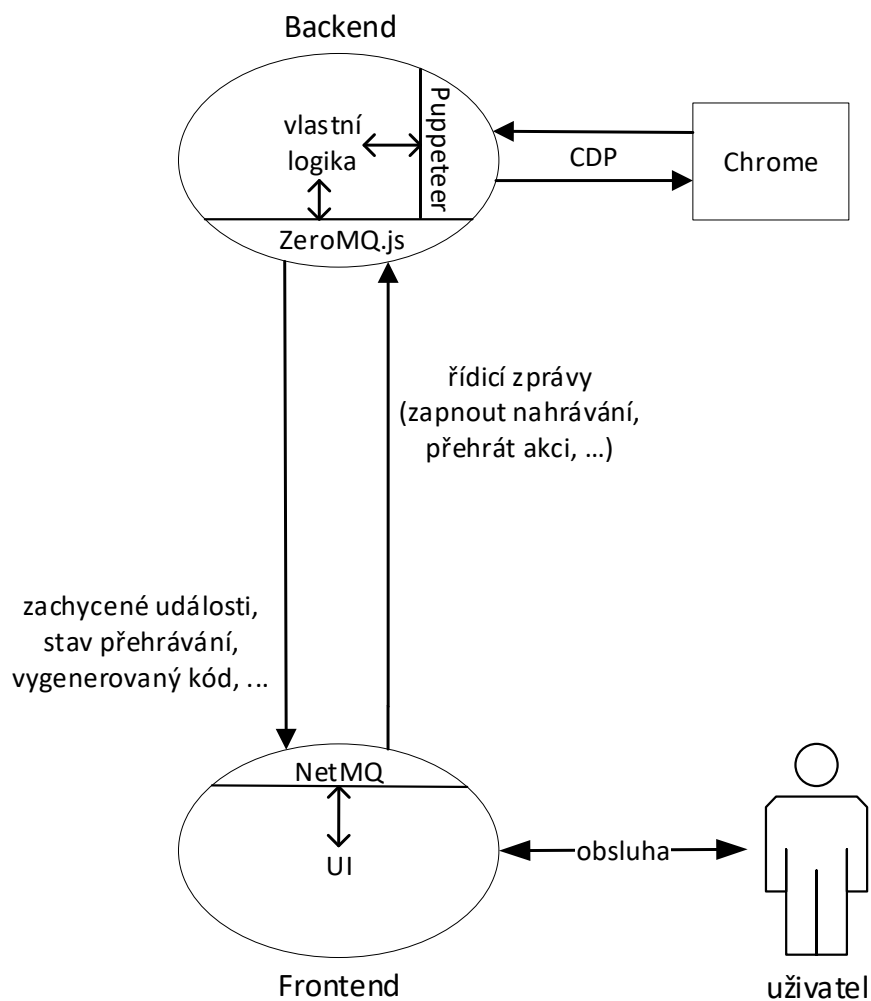
- Existence mnoha nativních portů, případně language bindingů pro spoustu jazyků [75]. Pro nás zásadní je existence portu NetMQ [73] pro C#² a bindingu ZeroMQ.js [74] pro JavaScript.
- ZeroMQ nevyžaduje message broker, vrstvu mezi frontendem a backendem zpracovávající zprávy [76] viz A.9.
- Jednoduché API umožňující přímé odeslání zpráv jako textových řetězců viz A.9.

¹ WPF (Windows Presentation Foundation) vyžadují ke svému běhu .NET Framework 3.0, jeho otevřená implementace Mono sice na svém webu deklaruje kompatibilitu s verzí 4.7, explicitně však uvádí, že WPF nepodporuje [66].

UWP (Universal Windows Platform) je z hlediska osobních počítačů dle oficiální dokumentace [67] kompatibilní pouze s Windows 10.

² Existuje i binding clrzmq4 [71], dokumentace [72] ale doporučuje použít NetMQ.

Design řešení včetně jeho částí a komunikace mezi frontendem a backendem ilustruje následující diagram.



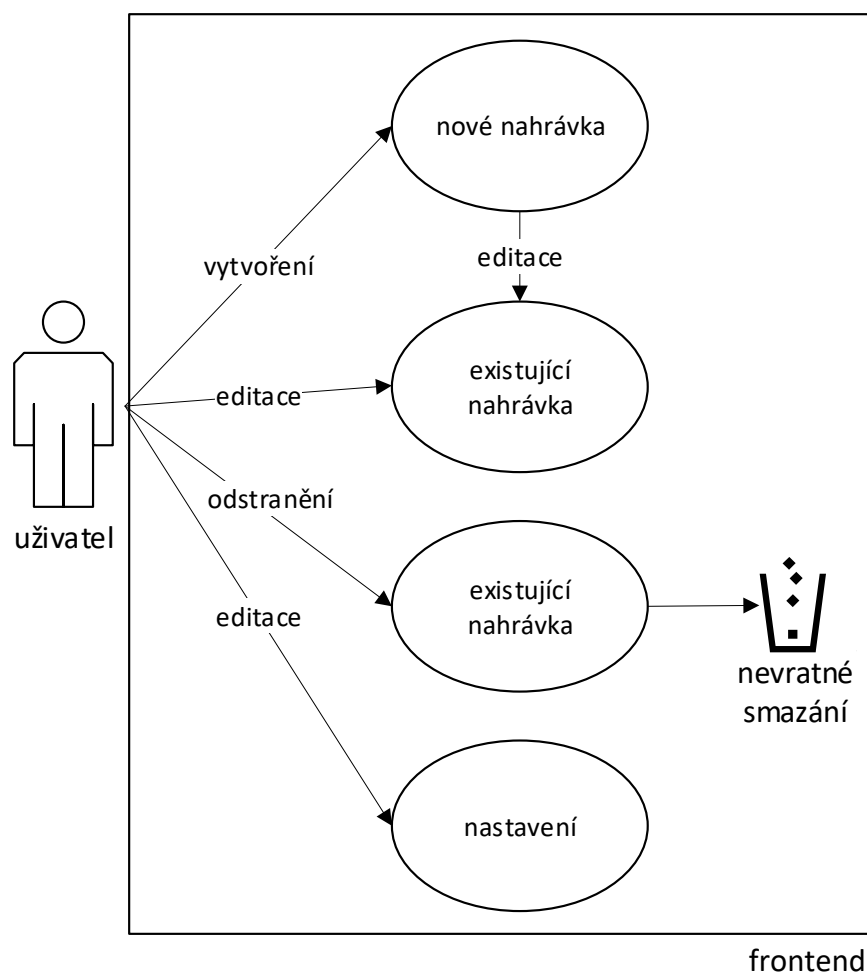
Obrázek 14: Diagram řešení

4.2 Scénáře použití

V této části bude následovat několik diagramů, které ilustrují možnosti uživatelů pro využití řešení.

Hlavní okno

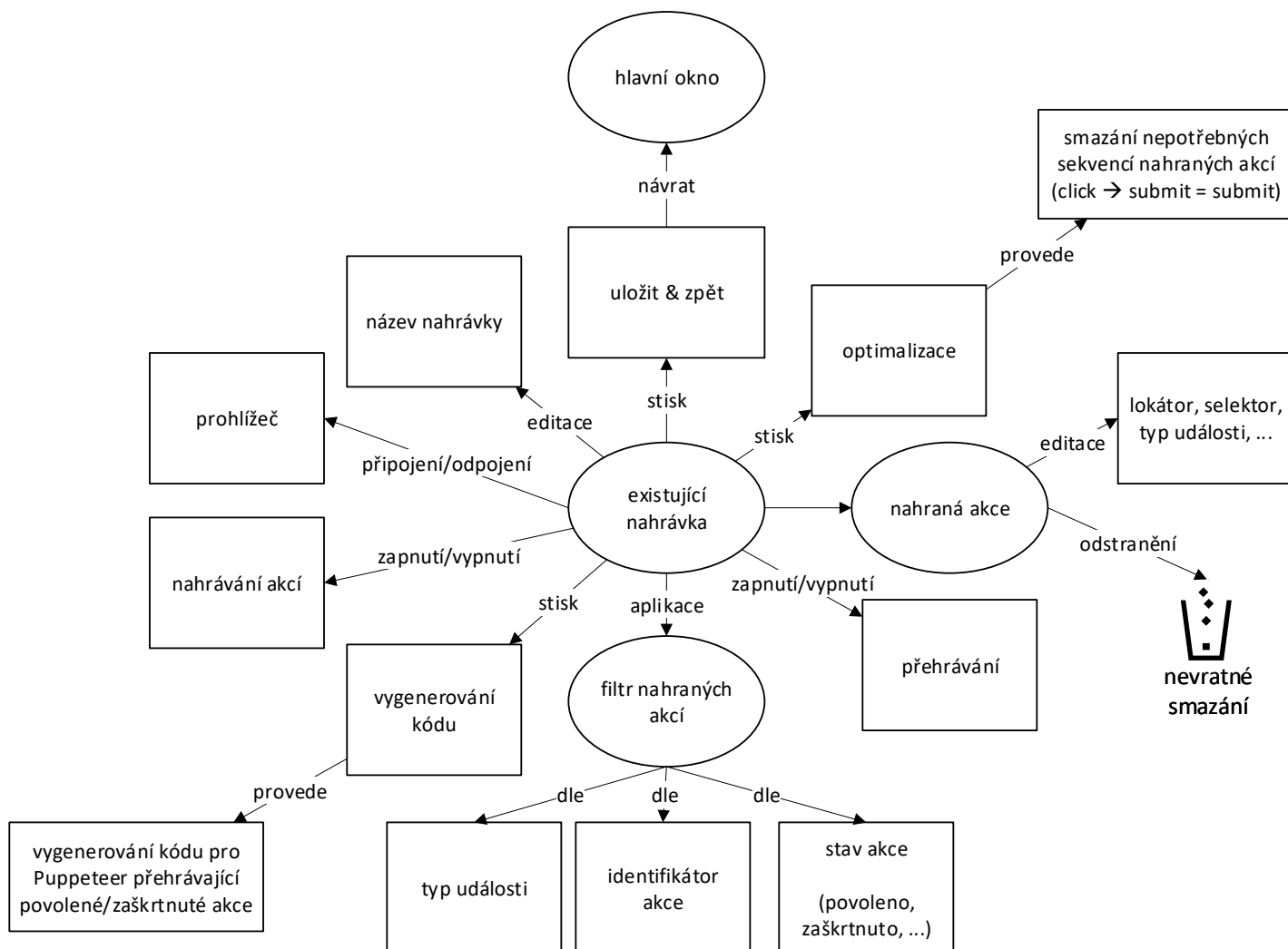
Hlavní okno reprezentuje stav ihned po spuštění.



Obrázek 15: Diagram hlavního okna

Editace existujícího nahrávky

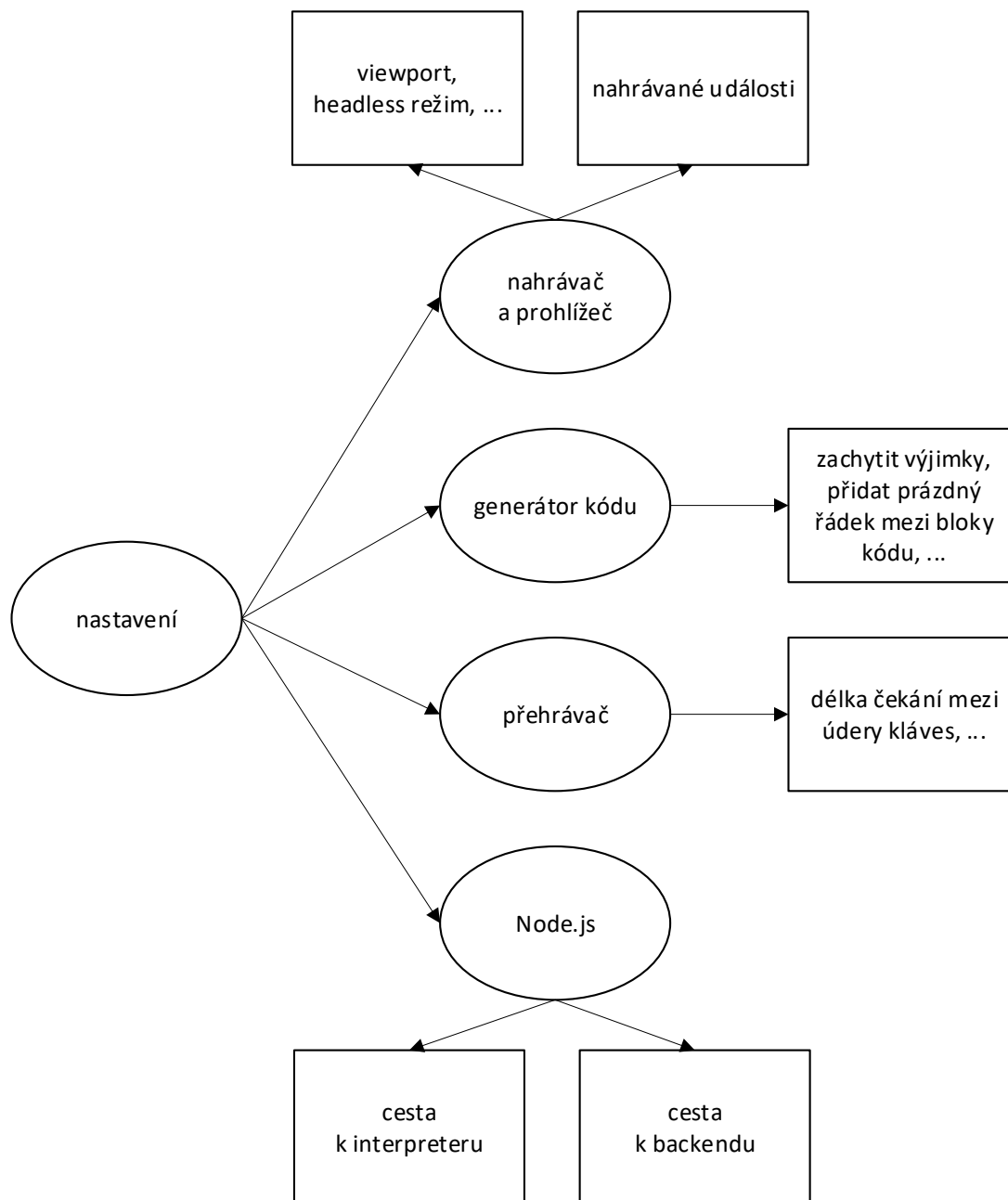
Zde rozvedeme konkrétní možnosti pro editaci existující nahrávky.



Obrázek 16: Diagram editace existující nahrávky

Editace nastavení

Nyní si ukážeme některé konkrétní možnosti editace nastavení.



Obrázek 17: Diagram editace nastavení

4.3 Porovnání funkcionality oproti konkurenčním řešením

Ještě předtím než začneme porovnávat bychom rádi upozornili na zásadní odlišující faktor vlastního řešení, a sice využití Puppeteeru. Ostatní řešení zmíněná v této práci využívala Selenium, případně byla proprietární. (Výjimkou je Headless Recorder, ten je postavený nad Puppeteerem, ale neumí přehrávání – nepovažujeme ho za plnohodnotné řešení.)

Události viewportu

Událost	Selenium IDE	Katalon Recorder	Ranorex Recorder	Headless Recorder	Vlastní řešení
click	✓	✓	✓	✓	✓
dblclick	✓	✓	✓	vygeneruje kód pro 2x click	✓
input	✓	✓	✓	✓	✓
change	✓	✓	✓	✓	✓
select	✗	✗	provádí relativní pohyb kurzoru simulující výběr textu	✗	✓
submit	✓	✓	provede akci spouštějící submit	provede akci spouštějící submit	✓
scroll	✓	✗	✓	✗	✓
copy, paste	pracuje s vlastní interní schránkou	pracuje s vlastní interní schránkou	✓	✗	✓
mouseover	pomocí kontextové nabídky	✗	simulace pohybu kurzoru	✗	podržením klávesy 'h'

Tabulka 10: Porovnání podporovaných událostí viewportu napříč řešeními

Události okna prohlížeče

Událost	Selenium IDE	Katalon Recorder	Ranorex Recorder	Headless Recorder	Vlastní řešení
otevření nového tabu	implicitně s akcí elementu	implicitně s akcí elementu	✓	✗	✓
uzavření existujícího tabu	✓	✓	✓	✗	✗
změna URL adresy	ručním přidáním akce open	ručním přidáním akce open	✓	✗	✓
změna aktivního tabu	✓	✓	✓	✗	detekováno s následující událostí

Tabulka 11: Porovnání podporovaných událostí okna prohlížeče napříč řešeními

Identifikátory elementů

Typ identifikátoru	Selenium IDE	Katalon Recorder	Ranorex Recorder	Headless Recorder	Vlastní řešení
selektory	✓	✗	využívá proprietárně rozšířený xpath identifikující i mimo viewport	✓	✓
lokátory	✓	pouze id, name, xpath		✗	✓

Tabulka 12: Porovnání podporovaných identifikátorů elementů napříč řešeními

Čekání na

Čekání na	Selenium IDE	Katalon Recorder	Ranorex Recorder	Headless Recorder	Vlastní řešení
existenci elementu	✓	✓	✓	✓	✓
hodnotu atributu elementu	✗	✓	✓	✗	✗
možnost editace elementu	✓	✓	✓	✗	✗
dokončení navigace	implicitní čekání na zavolání události 'load'	implicitní čekání na zavolání události 'load'	✓	implicitní čekání na zavolání události 'load'	✓
viditelnost elementu	✓	✓	✓	✗	✗

Tabulka 13: Porovnání podporovaných druhů čekání napříč řešeními

Možnosti exportu

Technologie	Selenium IDE	Katalon Recorder	Ranorex Recorder	Headless Recorder	Vlastní řešení
C# (MSTest)	✗	✓	✗	✗	✗
C# (NUnit)	✓	✓	✗	✗	✗
C# (xUnit)	✓	✗	✗	✗	✗
Java (JUnit)	✓	✓	✗	✗	✗
JavaScript (Mocha)	✓	✗	✗	✗	✗
JavaScript (New Relic Synthetics)	✗	✓	✗	✗	✗
JavaScript (Playwright)	✗	✗	✗	✓	✗

JavaScript (Puppeteer)	✗	✗	✗	✓	✓
JavaScript (WebDriver.io)	✗	✓	✗	✗	✗
JSON	✗	✓	✗	✗	✗
Katalon Studio	✗	✓	✗	✗	✗
mezikód pro výrobu nových formátů	✗	✓	✗	✗	✗
Python (App- Dynamics)	✗	✓	✗	✗	✗
Python (pytest)	✓	✗	✗	✗	✗
Python 2 (unittest)	✗	✓	✗	✗	✗
Robot Framework	✗	✓	✗	✗	✗
Ruby RSpec	✓	✓	✗	✗	✗
samostatný spustitelný soubor	✗	✗	✓	✗	✗
TypeScript (Protractor)	✗	✓	✗	✗	✗
XML	✗	✓	✗	✗	✗

Tabulka 14: Porovnání podpory generování kódu napříč řešeními

Další pokročilé funkce

Funkce	Selenium IDE	Katalon Recorder	Ranorex Recorder	Headless Recorder	Vlastní řešení
větvení (podmínky)	if, else if, else	if, else if, else	možnost propojení s C#	✗	✗
cykly	do, times, while, foreach	while	možnost propojení s C#	✗	✗
zvýraznění identifikovaného elementu uvnitř viewportu	✓	✓	dokonce i streamování aktuální podoby elementu	✗	✗
změna elementu existující akce klikem uvnitř viewportu	✓	✓	pomocí sofistikované součásti Ranorex Spy	✗	ruční změna nebo nahrání nové akce
ruční přidání akce	✓	✓	✓	✗	✗
nastavení velikosti viewportu	✓	✗	✓	✗	bez dynamických změn během nahrávání
volitelné zpomalení přehrávání	✓	✓	✓	✗	✓
připojení na vzdálený prohlížeč	✗	✗	pouze přehrávání přes RDP	✗	✓

Tabulka 15: Porovnání dalších pokročilých funkcí napříč řešeními

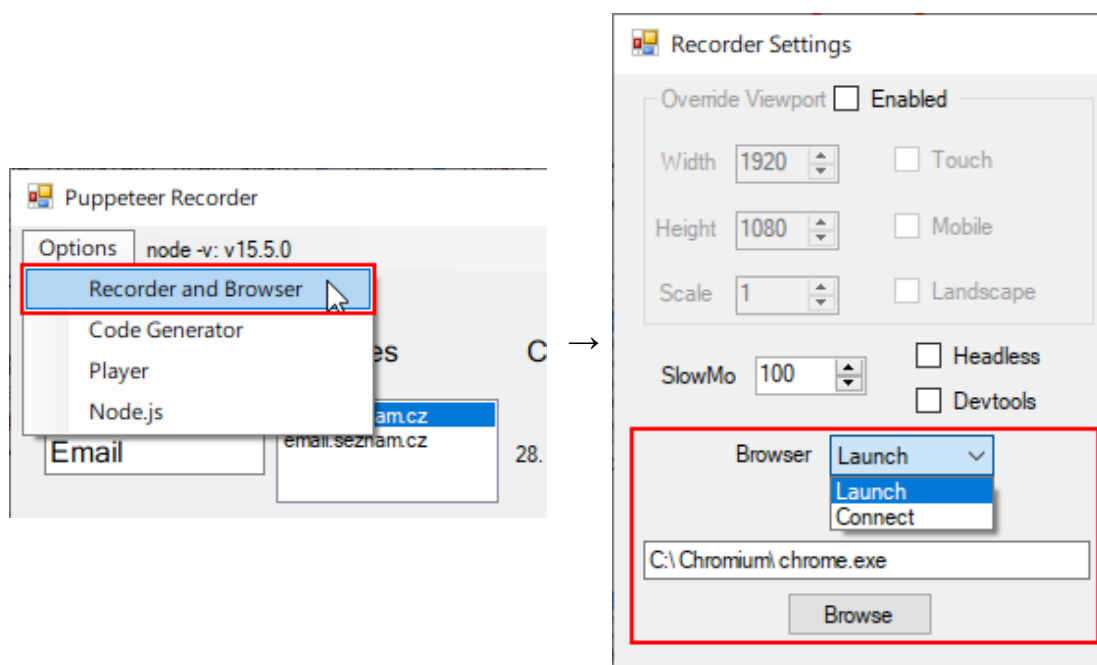
4.4 Tutoriály

Nyní se podíváme na UI frontendové části přes kterou se naše řešení ovládá. Využití uživatelského rozhraní bude demonstrováno několika tutoriály.

Tutoriály jsou vytvořené tak, aby ukazovaly základní funkcionalitu řešení, případně vysvětlovali neintuitivní části UI. Velká část zřejmých komponent UI je vynechána pro přehlednost v dokumentaci.

4.4.1 Prvotní spuštění

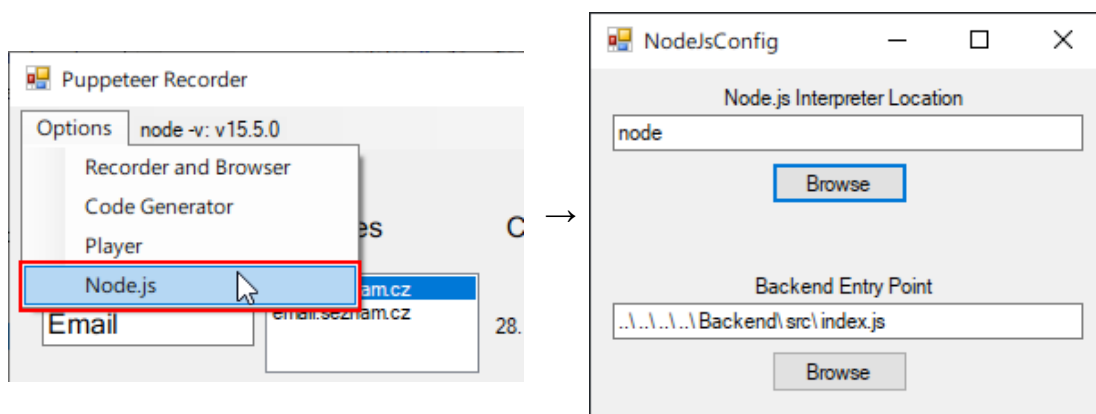
Pro správnou funkci je nutné určit typ připojení ke Chromu, nastavit cestu k interpretu Node.js pokud není umístěna v proměnné Path proměnného prostředí. Jestliže byla změněna relativní cesta backendu vůči frontendu, tak je nutné nastavit novou cestu i k backendu. Začneme připojením Chromu.



Obrázek 18: Nastavení připojení ke Chromu

Jak je patrné z obr. 18, na výběr máme ze dvou typů připojení. „Launch“ spustí novou lokální instanci Chromu, volitelně vyžaduje zadání cesty ke spustitelnému souboru prohlížeče, její zadání je nutné pokud se spustitelný soubor Chromu nenachází ve standardním umístění. Druhý typ připojení označený jako „Connect“ se připojí k existující vzdálené instanci Chromu, více o tomto druhu připojení viz tutoriál 4.4.3 Připojení ke vzdálenému Chromu.

Poslední dvě zbývající cesty pro backend a Node.js se nastavují ve společném dialogovém okně.



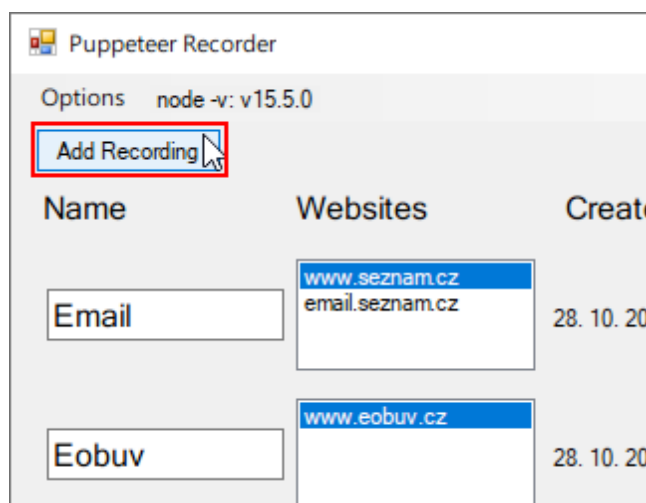
Obrázek 19: Nastavení cesty k interpreteru Node.js a backendu

Obě vyplněné cesty jsou výchozí hodnotou nastavení. Všimněme si, že jako cesta k interpreteru Node.js stačí uvést pouze „node“ pokud je jeho umístění v proměnné Path proměnného prostředí.

Cesta k backendu je popsána relativně vůči umístění spustitelného souboru frontendové části. Pokud nebyla upravena struktura projektu, cesta k backendu je správná.

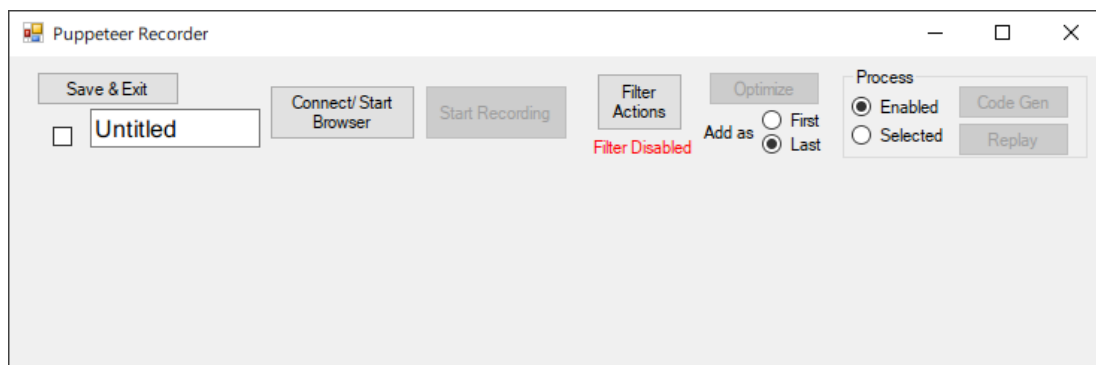
4.4.2 Nová nahrávka

Zkusíme si vytvořit jednoduchou nahrávku na které si ukážeme základní součásti editačního UI pro nahrávky.



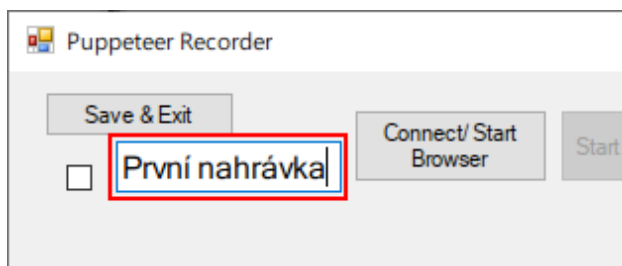
Obrázek 20: Vytvoření nové nahrávky

Po stisknu zvýrazněného tlačítka na obr. 20 se změní podoba okna pro nahrávání a přehrávání akcí, okno bude vypadat takto:



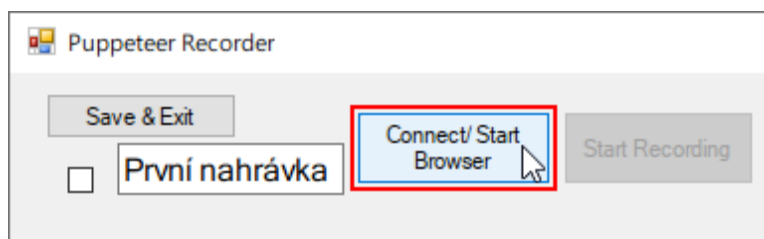
Obrázek 21: Editační UI pro nahrávky

Nyní máme prázdnou nahrávku, naším cílem bude nahrát několik akcí, přehrát je a zobrazit si jejich kód pro Puppeteer. Předtím však změníme jméno nahrávky z „Untitled“ na „První nahrávka“.



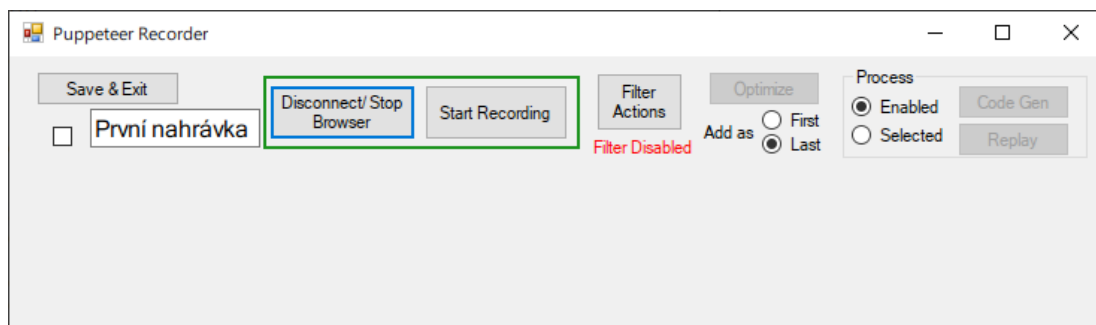
Obrázek 22: Změna jména nahrávky

Pokud jsme prošli tutoriál 4.4.1 Prvotní spuštění, měli bychom mít vše správně nastavené a Chrome v režimu spuštění lokální instance. Spustíme backend a skrz něj i Chrome tlačítkem „Connect/Start Browser“.



Obrázek 23: Spouštění backendu a Chromu

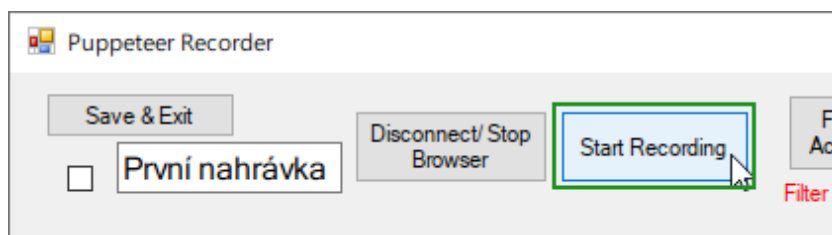
Měla by se nám otevřít nová instance Chromu s jedním prázdným otevřeným oknem. UI by mělo povolit zakázané tlačítko „Start Recording“ a vypadat takto:



Obrázek 24: Editační UI po spuštění backendu a prohlížeče

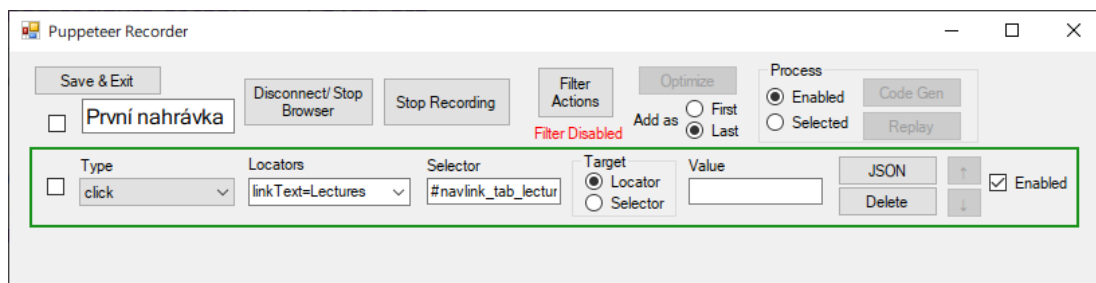
Pokud UI tak nevypadá a Chrome se neotevřel, objevila zpráva s chybou, ta vždy popisuje konkrétní problém včetně kroků, které je nutné podniknout pro její odstranění, chyba by se ale po správném provedení kroků v tutoriálu 4.4.1 Prvotní spuštění objevit neměla.

Stiskem tlačítka „Start Recording“ spustíme nahrávání akcí provedených s otevřenou instancí Chromu. Pro seznam podporovaných akcí odkážeme čtenáře do 4.3 Porovnání funkcionality oproti konkurenčním řešením.



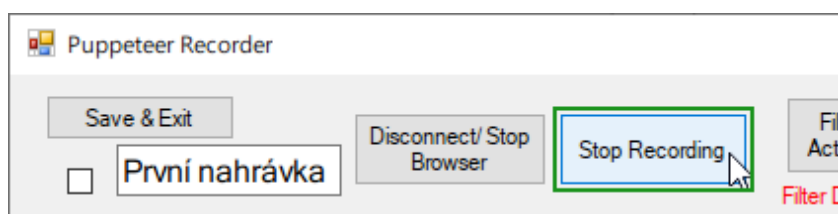
Obrázek 25: Zapnutí nahrávání akcí

Nahrajme si libovolný počet akcí. Nahrání akce poznáme přidáním odpovídajícího řádku do UI.



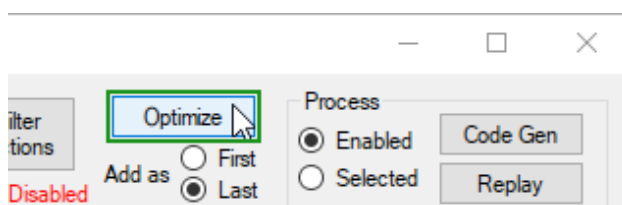
Obrázek 26: Nahrání první akce

Pro ukončení nahrávání akcí klikneme na tlačítko „Stop Recording“.



Obrázek 27: Vypnutí nahrávání akcí

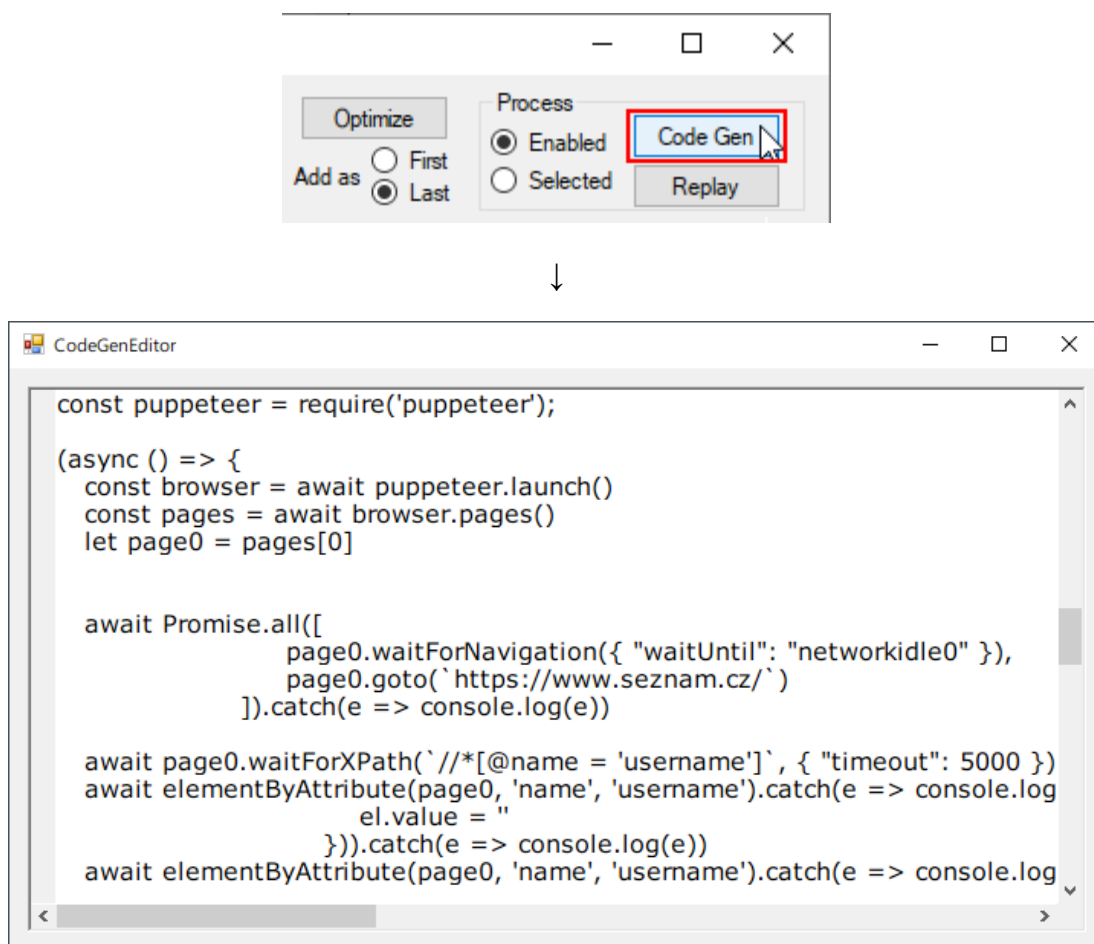
Nyní nás již dělí jediný krok od generování kódu pro Puppeteer a přehrávání akcí, tím krokem je optimalizace akcí. Při nahrávání akcí často nahrajeme spoustu akcí z nichž značná část je pro přehrávání nepodstatná¹. Pro smazání snadno popsatelných nadbytečných akcí je zde přítomno tlačítko „Optimize“. Klikem na toto tlačítko nyní optimalizaci provedeme.



Obrázek 28: Optimalizace akcí

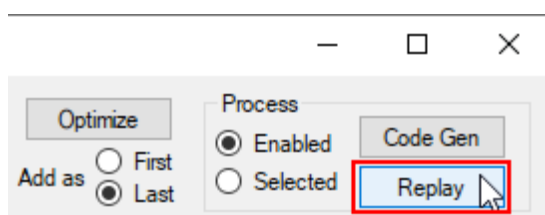
¹ Uvedeme příklad jednoduše popsatelné nepodstatné akce: Při vyplňování textového políčka se nahrají události „input“, po vyplnění a opuštění políčka se nahraje událost „change“. Pro přehrávání nám stačí pouze událost „change“. Optimalizace v tomto případě odstraní všechny události „input“. Uvědomme si však, že řešením není nahrávat pouze události „change“ a úplně ignorovat události „input“. V tomto případě by se mohlo stát, že uživatel vyplní textové políčko, neopustí ho, vypne nahrávání a akce s vyplněním políčka nebude nahrána.

Aktuálně nám již nic nebrání vyzkoušet si vygenerovat kód a přehrát akce.



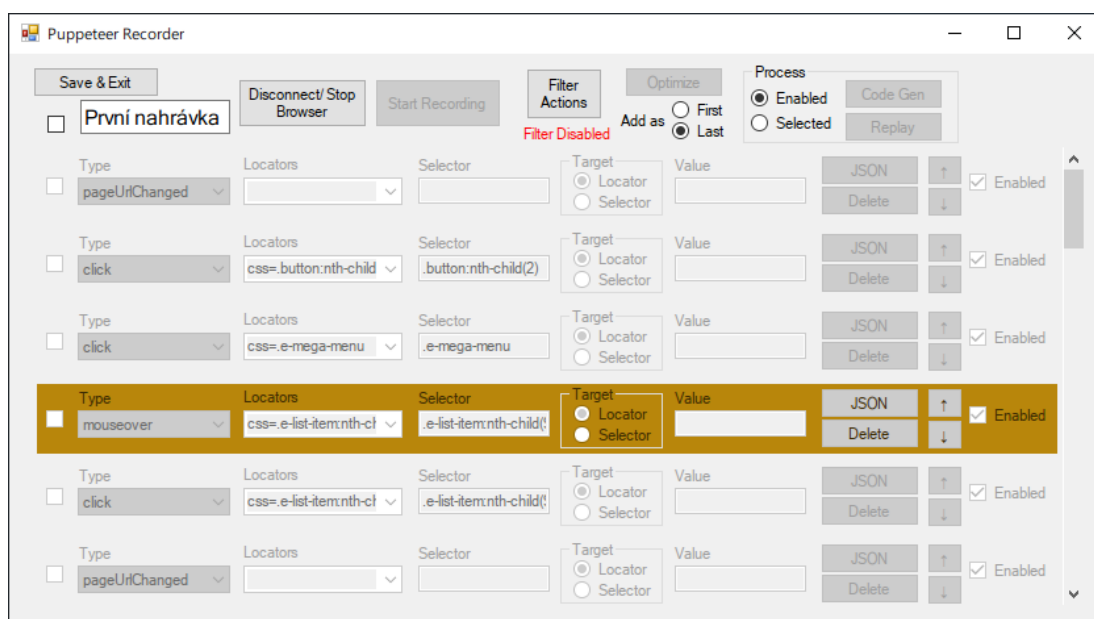
Obrázek 29: Zobrazení kódu pro Puppeteer odpovídající nahraným akcím

Akce přehrajeme kliknutím na tlačítko „Replay“.



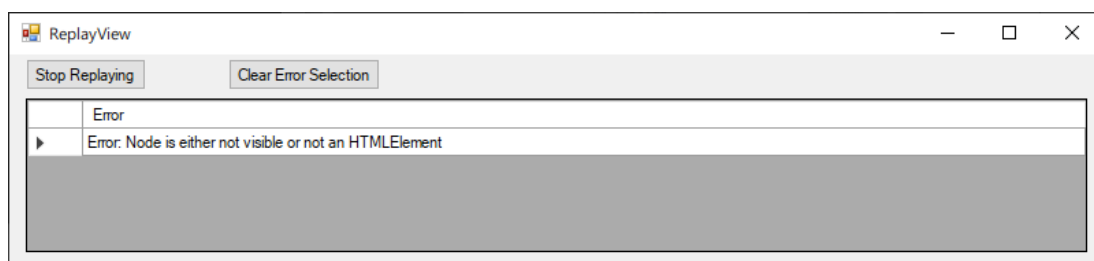
Obrázek 30: Zapnutí přehrávání akcí

Po kliku se nám zvýrazní právě přehrávaná akce.



Obrázek 31: Spuštění přehrávání akcí

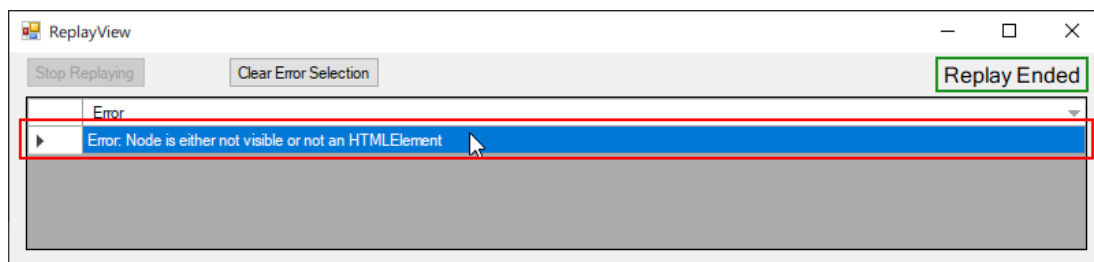
Otevře se i nové okno popisující stav probíhajícího přehrávání. Zde se zobrazí případné chyby¹ při přehrávání akcí, probíhající přehrávání je možné tlačítkem „Stop Replaying“ náhle ukončit.



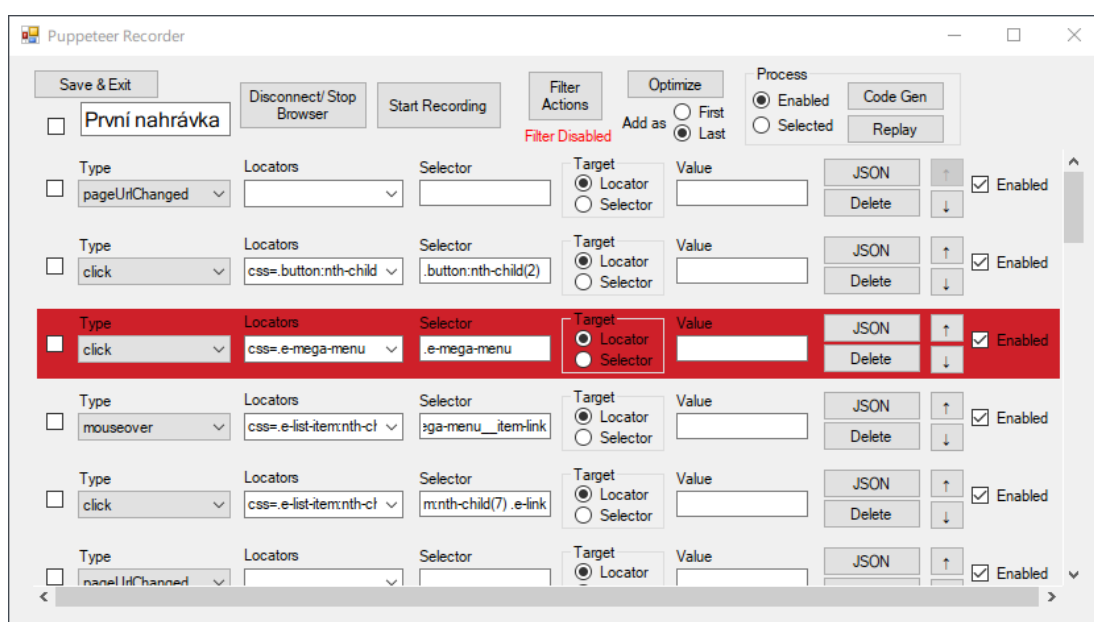
Obrázek 32: Okno popisující stav spuštěného přehrávání

¹ Chybami rozumíme neexistenci elementu a existenci více elementů pro daný identifikátor.

Při kliknutí na zprávu oznamující chybu se nám v editačním UI zvýrazní akce, která chybu vyvolala.



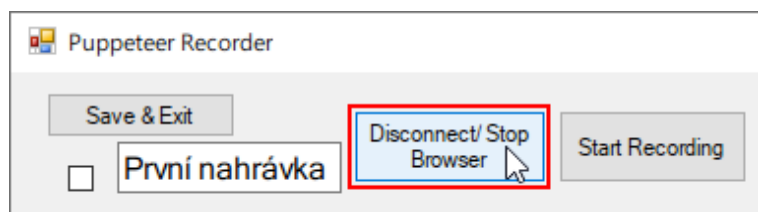
(a)



Obrázek 33: Zvýraznění akce, která vyvolala chybu

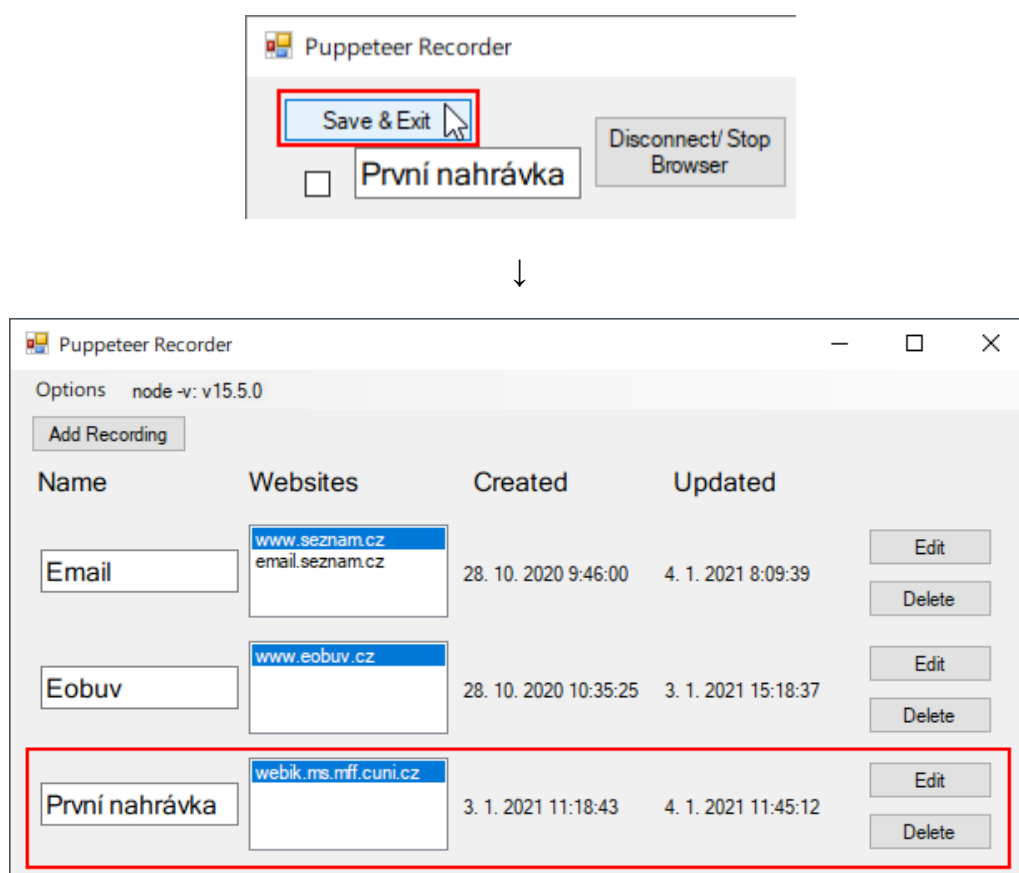
Po dokončení přehrávání se nám v okně popisující přehrávání objeví zpráva „Replay Ended“, ta je zvýrazněna zeleným rámečkem na obr. 33(a).

Přesuneme se od přehrávání – vraťme se nyní do hlavního spouštěcího okna frontendu. Okno popisující přehrávání zavřeme, odpojíme se od prohlížeče a vypneme backend.



Obrázek 34: Odpojení prohlížeče a vypnutí backendu

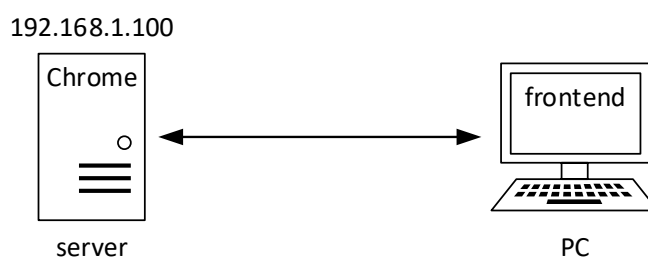
Nakonec uložíme nahrávku a vrátíme se do seznamu všech uložených nahrávek.



Obrázek 35: Uložení nahrávky a návrat do seznamu všech uložených nahrávek

4.4.3 Připojení ke vzdálenému Chromu

Pro účely tohoto tutoriálu je nutné mít připravenou jednu nahrávku a dva počítače. Předpokládejme následující schéma zapojení.



Obrázek 36: Schéma zapojení počítačů

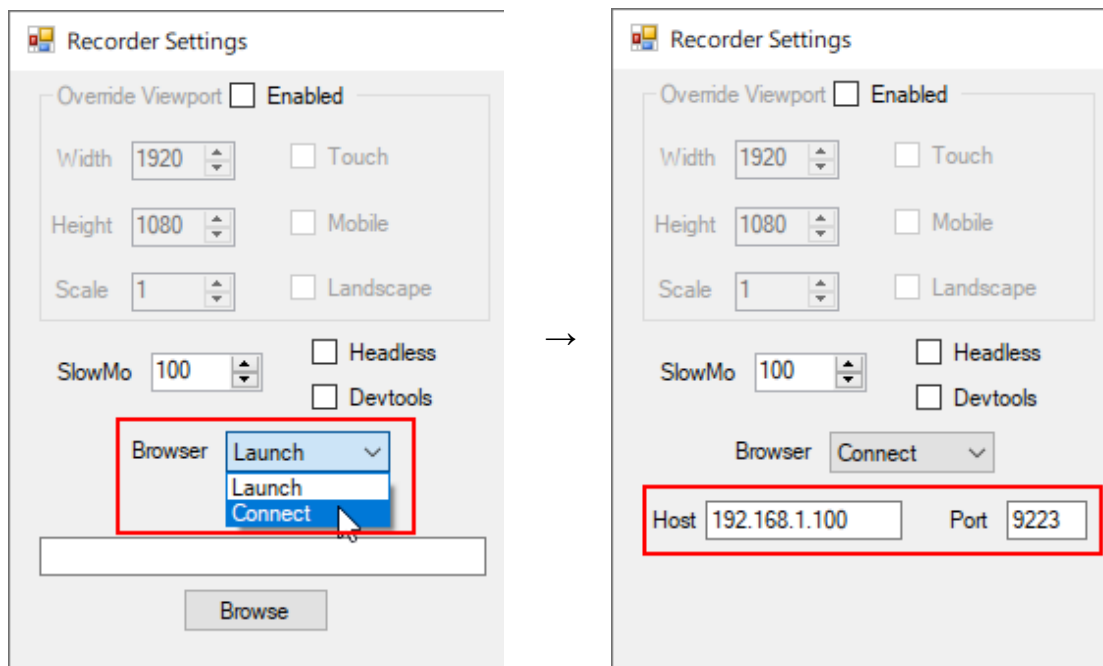
Na serveru spustíme Chrome s parametrem `--remote-debugging-port=9222`, s tímto parametrem běží ve Chromu CDP na portu 9222. Na CDP takto spuštěné instance Chromu je možné se připojit pouze z localhost [9], abychom CDP zpřístupnili

vzdáleně, musíme přesměrovat nějaký jiný vzdáleně přístupný port na port 9222. Tohoto je možné docílit spuštěním následujícího příkazu na serveru [77].

```
1 ssh -L 0.0.0.0:9223:localhost:9222 localhost -N
```

Ukázka kódu 23: Přesměrování vzdáleně přístupného portu 9223 na port 9222

Nyní stačí nastavit ve frontendové části připojení ke Chromu na „Connect“, zadat korektní IP adresu s portem.

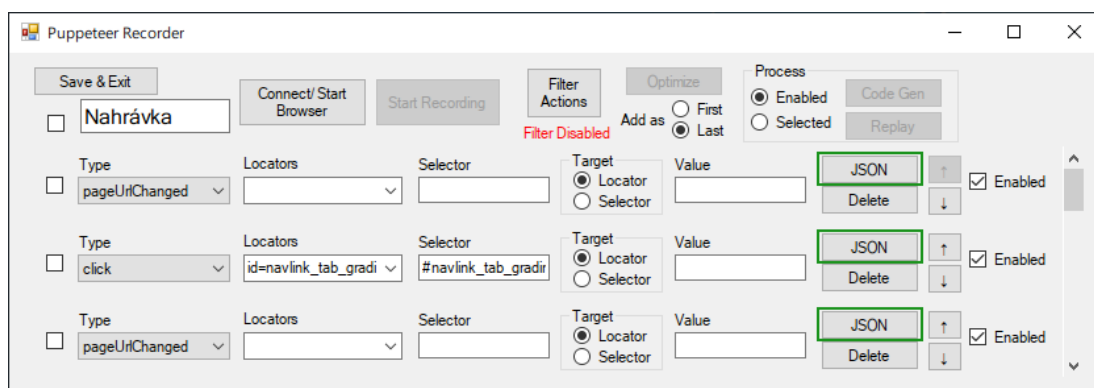


Obrázek 37: Nastavení připojení na „Connect“ s vyplněním IP adresy a portu

Můžeme si vyzkoušet přehrát připravenou nahrávku. Připojení ke Chromu a spuštění backendu tlačítkem „Connect/Start Browser“ nespustí novou instanci Chromu, ale připojí se k již existující na serveru. Přehrávání nahrávky se odehraje na serveru.

4.4.4 Tlačítko JSON

Součástí každé nahrané akce je tlačítko „JSON“.



Obrázek 38: Tlačítko JSON jako součást každé akce

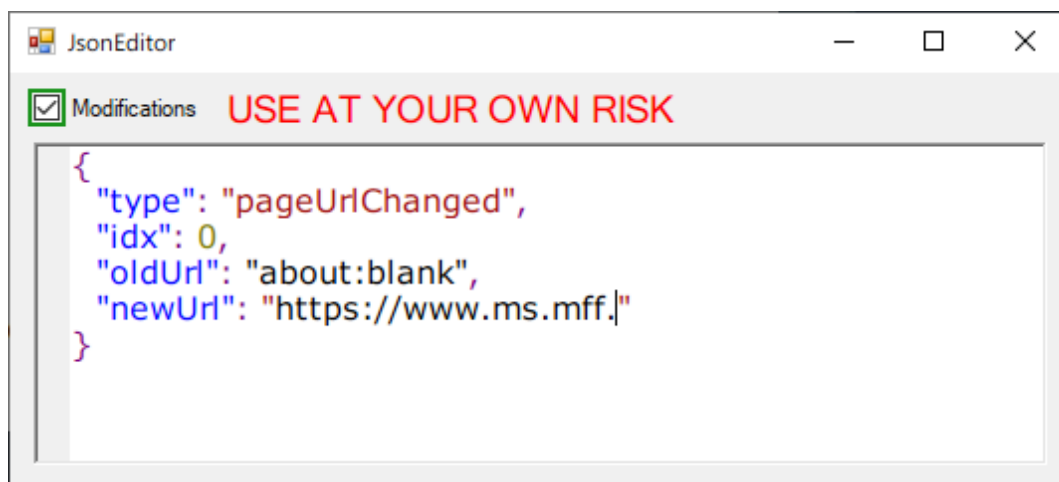
Po kliku na toto tlačítko se zobrazí textová nezpracovaná data ve formátu JSON popisující nahranou akci.



Obrázek 39: Data ve formátu JSON popisující nahranou akci „pageUrlChanged“

Obr. 39 popisuje podle parametru „type“ událost „pageUrlChanged“. Mezi další užitečné parametry patří „oldUrl“ a „newUrl“, zřejmě z jejich názvů: První parametr popisuje původní adresu URL, druhý pak novou adresu, která byla načtena. Ani jeden z těchto parametrů nemá vlastní UI viz obr. 38, a proto jediná možnost nahlédnutí na konkrétní hodnoty je přes data v JSONu.

Pokud bychom potřebovali nezpracovaná data v JSONu měnit, např. chceme načíst jinou adresu URL, máme k dispozici zaškrávací políčko s popiskem „Modifications“ (zvýrazněno zeleným rámečkem na obr. 39), které při zaškrtnutí povolí modifikace. Dovolujeme si ale upozornit, že takto provedené modifikace jsou prováděny na vlastní nebezpečí a při neplatné modifikaci je možné akci i znefunkčnit.

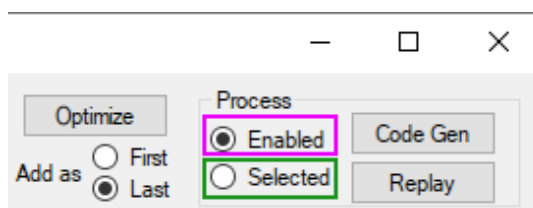


Obrázek 40: Modifikace nezpracovaných dat v JSONu

4.4.5 Výběr akcí ke zpracování

Některé nahrávky mohou obsahovat spousty akcí, ne vždy budeme chtít všechny přehrát. Dokonce můžou nastat situace, kdy budeme chtít výhradně pro debuggovací účely přehrát jednu jedinou akci – totéž platí kromě přehrávání i pro generování kódu. V tomto tutoriálu si ukážeme jak na to.

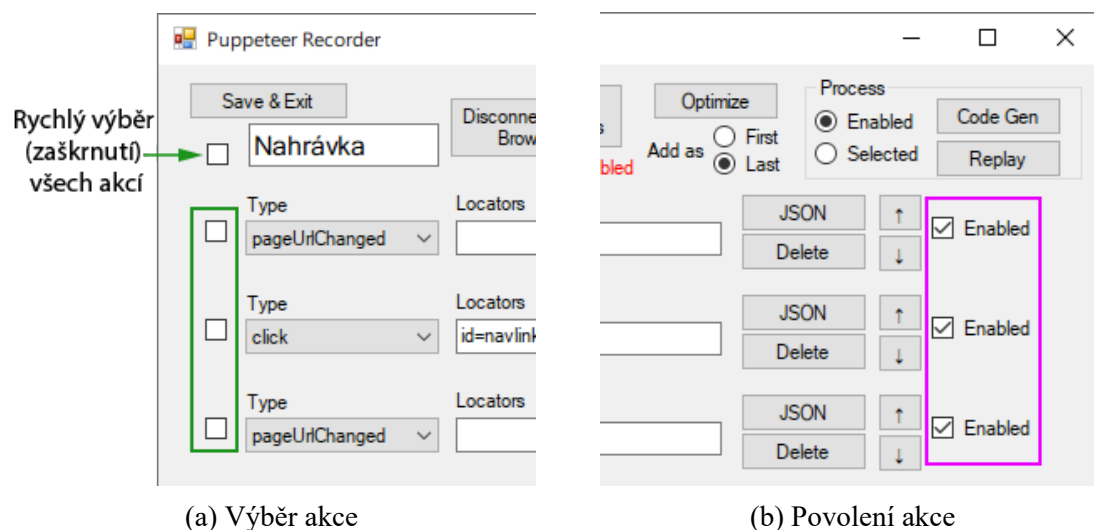
Přesuňme se do editačního UI libovolné nahrávky. Jak již bylo naznačeno v úvodním odstavci, výběr akcí je možný pro přehrávání i pro generování kódu.



Obrázek 41: Přepínače pro výběr akcí ke zpracování

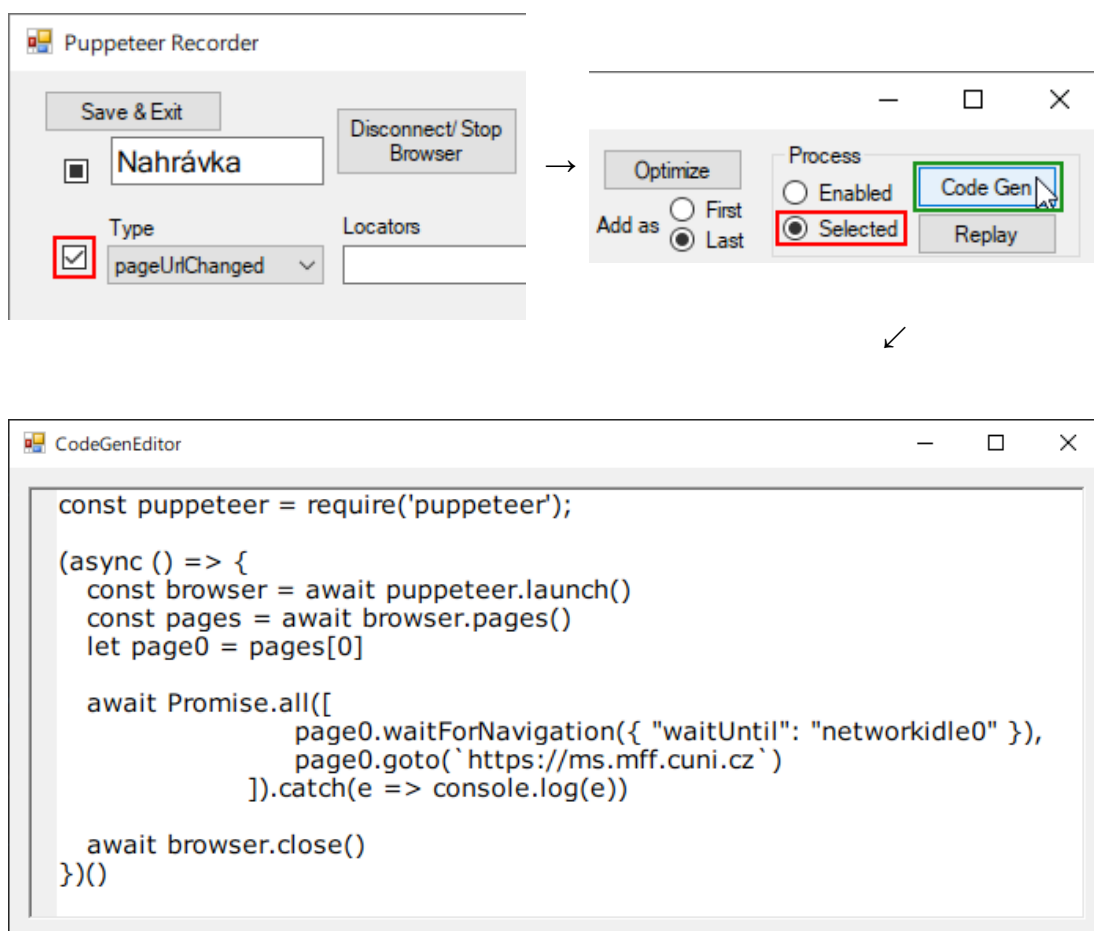
Obr. 41 nám ukazuje, že je možné zpracovat buďto akce povolené („Enabled“) nebo akce vybrané („Selected“).

Na následujícím obrázku jsou odpovídající zaškrtnuté políčka pro výběr a povolení akcí zvýrazněna stejnou barvou jako na obr. 41.



Obrázek 42: Zaškrtnuté políčka pro výběr a povolení akcí

Na závěr tohoto tutoriálu přikládáme ukázkou vygenerování kódu jedné akce typu „pageUrlChanged“.

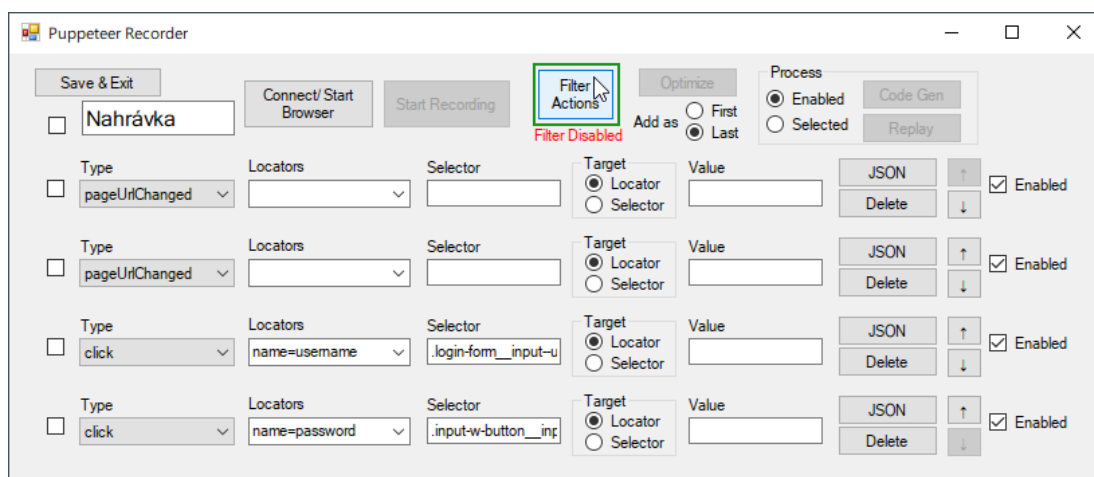


Obrázek 43: Vygenerování kódu jedné akce typu „pageUrlChanged“

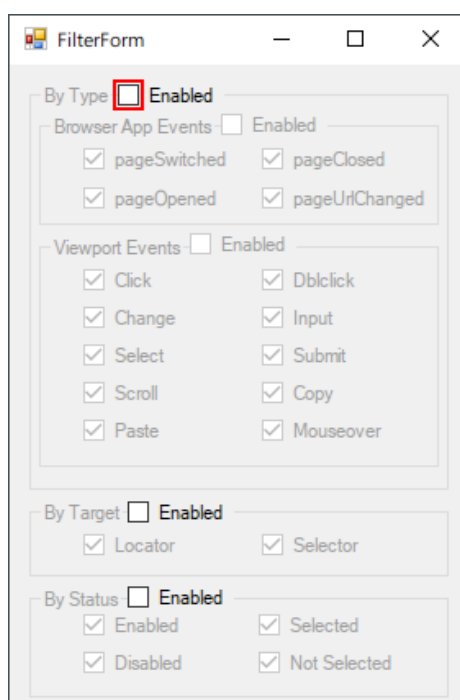
4.4.6 Filtr akcí

V horní části editačního UI je ještě jedno tlačítko, na které jsme se zatím v žádném tutoriálu nepodívali. Tlačítko má popisek „Filter Actions“, který nám napovídá, že jedná se o filtr. V našem případě umožňuje dočasně skrýt akce dle různých kritérií¹.

Přesuněme se do existující nahrávky a klikneme na tlačítko pro filtraci.



(a) Klik na tlačítko „Filter Actions“



(b) Okno pro aplikaci filtrů

Obrázek 44: Otevření okna pro aplikaci filtrů

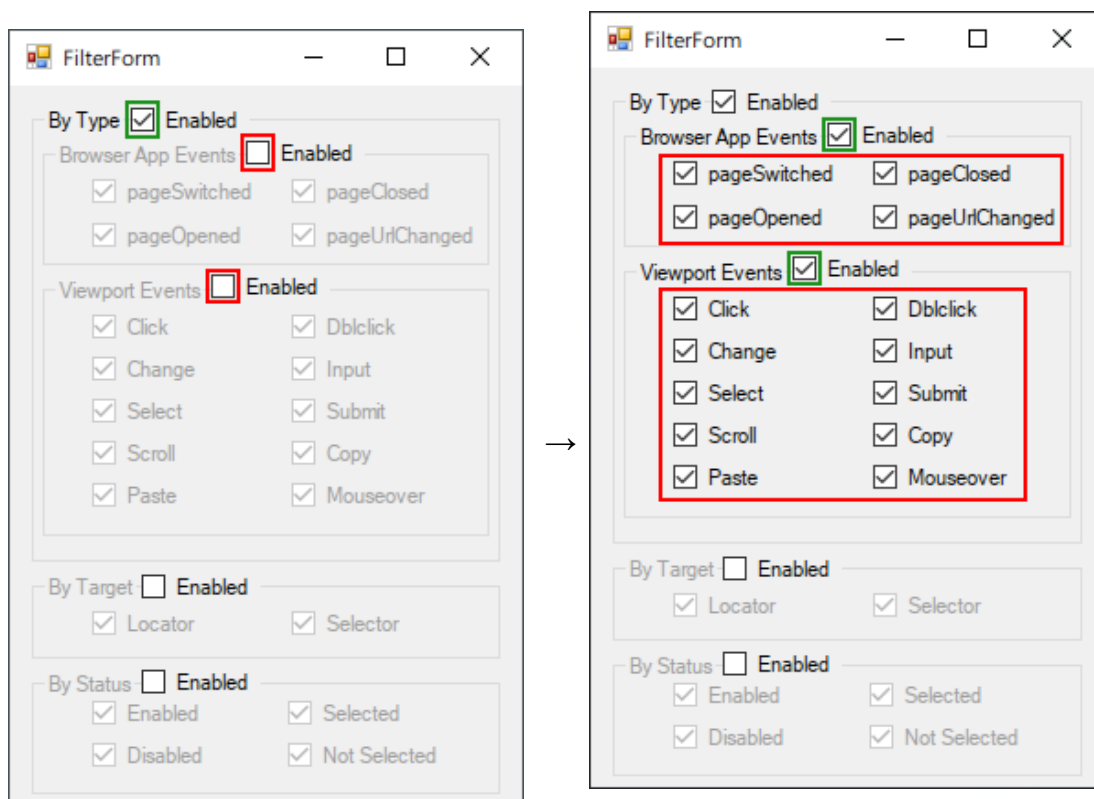
¹ Kritéria, která můžeme použít pro filtrování i současně:

- typ události (pageUrlChanged, click, scroll, ...)
- typ identifikátoru (lokátor, selektor)
- stav akce (povolena, vybraná)

Všimněme si, že na obr. 44(a) je pod stisknutým tlačítkem napsáno „Filter Disabled“, což značí neaktivitu filtru. Dále se podívejme, že na stejném obrázku se mezi čtyřmi nahranými akcemi nachází nejprve dvakrát za sebou akce typu „pageUrlChanged“ a potom dvakrát za sebou akce typu „click“. Filtr využijeme tak, že skryjeme poslední dvě akce.

Pro použití filtru nejprve musíme aktivovat filtr odpovídající kategorie. To se provádí zaškrtnutím políčka s popiskem „Enabled“. Políčko pro aktivaci filtru dle typu akce je zvýrazněno červeným rámečkem na obr. 44(b).

Tuto aktivaci filtru provedeme, poté zaškrtneme políčko s popiskem „Browser App Events“ a „Viewport Events“. První zmíněné políčko povolí filtrování akcí pocházejících z okna prohlížeče, druhé povolí filtrování akcí z viewportu.



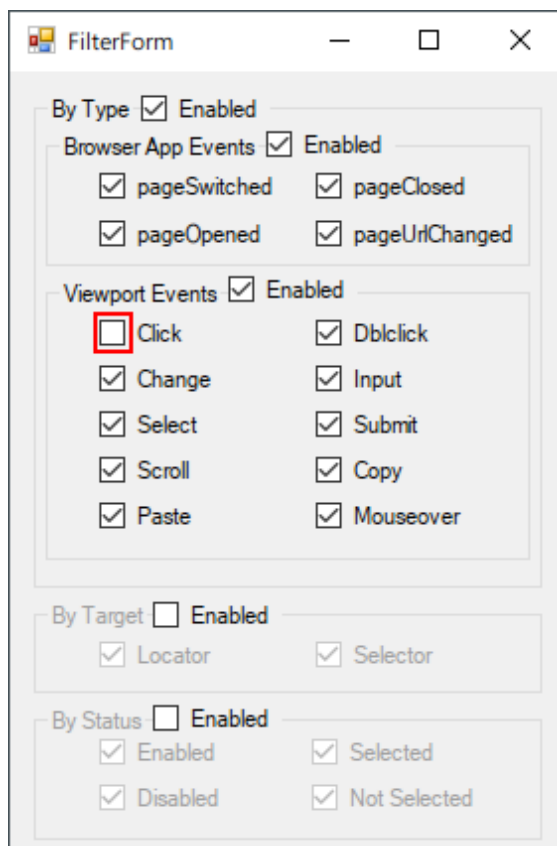
(a) Aktivace filtru dle typu akce

(b) Povolení filtrování akcí pocházejících z okna prohlížeče i z viewportu

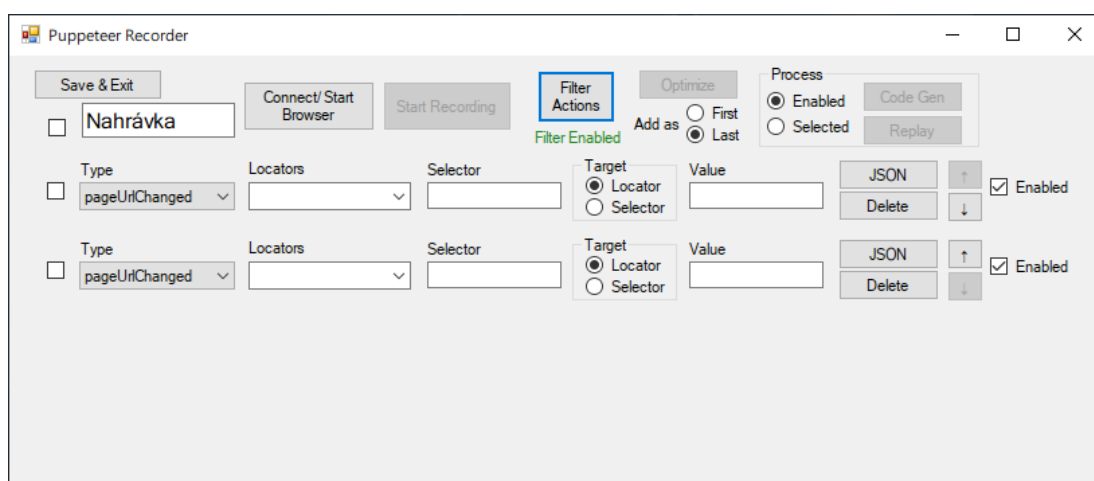
Obrázek 45: Aktivace filtru pro akce pocházející z okna prohlížeče a viewportu

Zaškrnutá políčka, která jsou obsažena v červeném rámečku na obr. 45(b) značí typy akcí, které budou zobrazeny v editačním UI.

Pro skrytí posledních dvou akcí typu „click“ stačí pouze odškrtnout políčko s popiskem „Click“.



(a) Odškrtnutí políčka s popiskem „Click“



(b) Editační UI se zapnutým filtrem

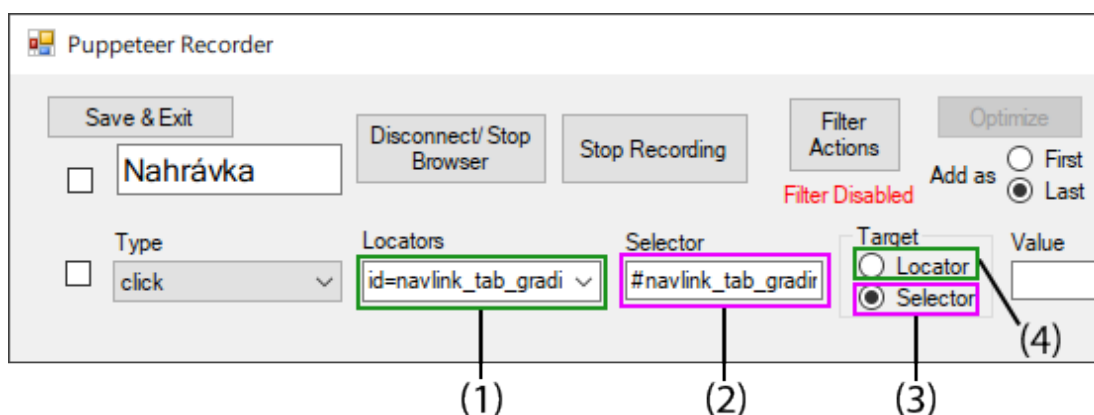
Obrázek 46: Skrytí dvou akcí typu „click“ pomocí filtru

Všimněme si, že na obr. 46(b) se nyní pod tlačítkem s popiskem „Filter Actions“ nachází text zelené barvy „Filter Enabled“ informující o zapnutí filtru.

4.4.7 Změna identifikátoru akce

Akce pocházející z viewportu prohlížeče jsou vždy vykonány nad elementem, který je potřeba identifikovat. K tomu nám slouží vhodný selektor nebo lokátor. V tomto tutoriálu si ukážeme jak pracovat se selektory a lokátory.

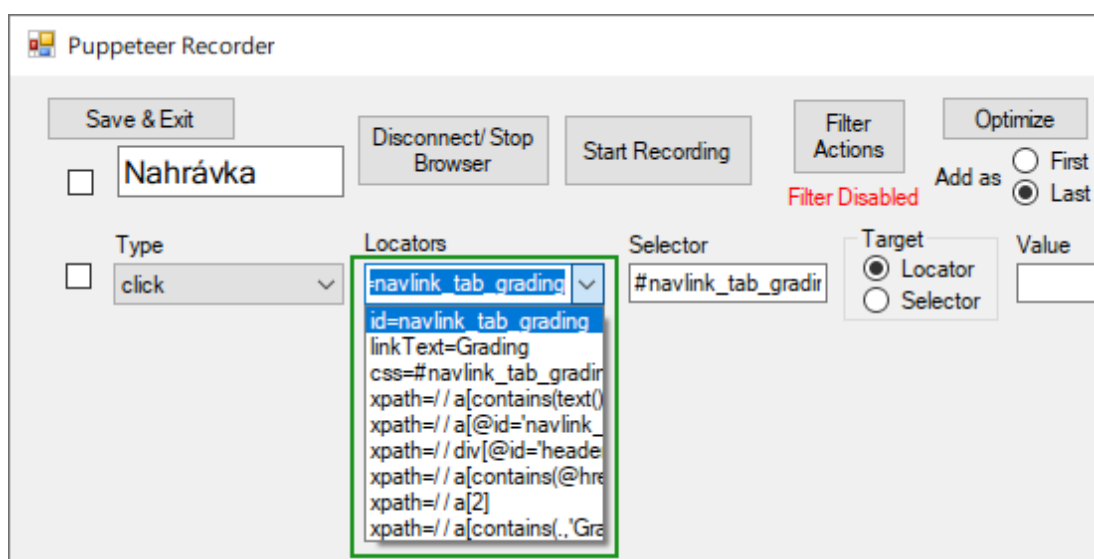
Přesuneme se opět do editačního UI existující nahrávky. Barvy rámečků na následujícím obrázku zobrazují vztahy mezi přepínači a políčky.



Obrázek 47: Vztahy mezi přepínači a políčky

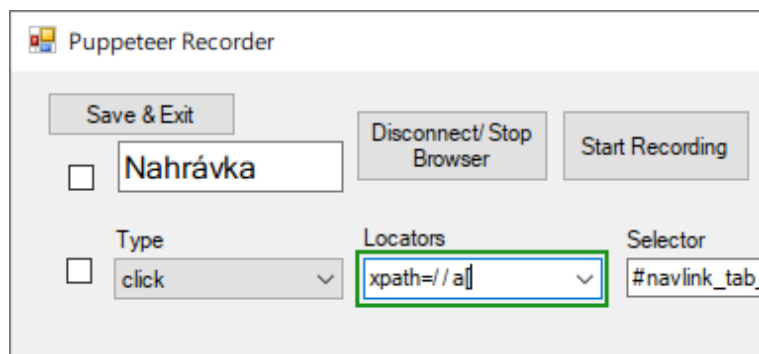
Na obr. 47 je vybrán přepínač (3), jako identifikátor se bude brát v potaz pouze hodnota selektoru (2).

Pokud bychom naopak chtěli použít lokátor jako identifikátor, stačí aby byl vybrán přepínač (4). V tomto případě se bude ignorovat hodnota (2) a použije se vybraný lokátor (1), povšimněme si, že se jedná o combo box – obvykle totiž existuje více různých lokátorů identifikujících element.



Obrázek 48: Seznam dostupných lokátorů pro konkrétní akci typu „click“

Pokud bychom chtěli použít vlastní lokátor, který není v seznamu mezi dostupnými, můžeme přepsat hodnotu combo boxu na libovolný vlastní lokátor.

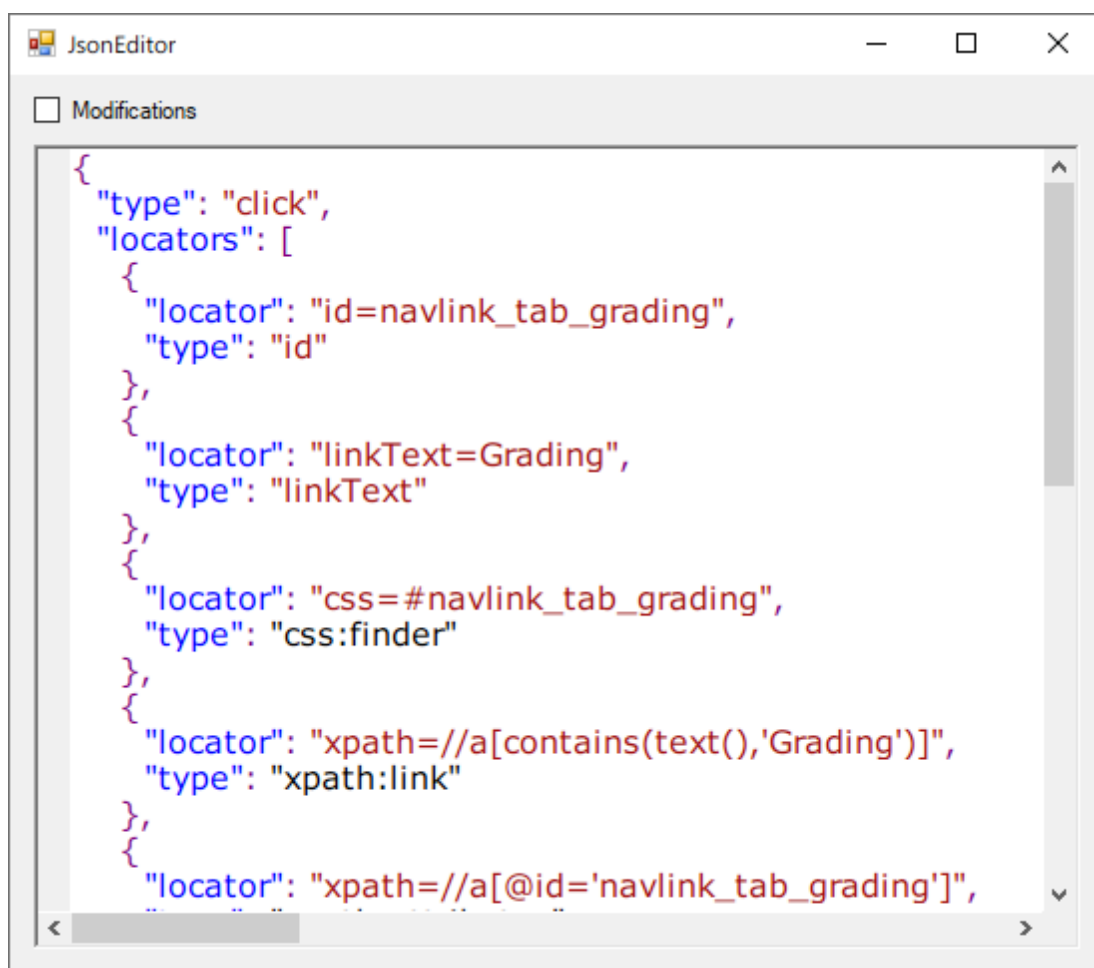


Obrázek 49: Vlastní lokátor

Seznam všech různých typů lokátorů je uveden v tab. 2 na straně 7.

Čtenáře, který pečlivě četl tutoriál 4.4.1 Prvotní spuštění jistě napadne, že další způsob přidání i většího množství vlastních lokátorů je pomocí tlačítka „JSON“.

Data ve formátu JSON popisující akci typu „click“ vypadají takto:



Obrázek 50: Data ve formátu JSON akce typu „click“

Po povolení modifikací můžeme přidat další lokátory.



Obrázek 51: Přidání nového lokátoru

4.5 Detaily implementace

Programátorská dokumentace

4.5.1 Frontend

4.5.2 Backend

5 Závěr

A Fragmenty kódu pro Puppeteer

Zde jsou umístěny části kódu pro Puppeteer, které nejsou považovány za nezbytné, případně jsou příliš dlouhé.

A.1 Zachycení událost z viewportu

```
1 import puppeteer from 'puppeteer'
2
3 const eventsToRecord = ['click', 'dblclick', 'change', 'input',
4                         'select', 'submit', 'scroll', 'copy',
5                         'paste', 'keydown', 'keyup', 'mouseover']
6
7 async function connectEvents(page) {
8   await page.evaluate(viewportEvents, eventsToRecord)
9   await page.evaluateOnNewDocument(viewportEvents, eventsToRecord)
10  await page.exposeFunction('captureViewportEvent',
11                             captureViewportEvent)
12 }
13
14 function viewportEvents(eventsToRecord) {
15   const listener = event =>
16     window.captureViewportEvent({type: event.type})
17
18   eventsToRecord.forEach(eventName =>
19     window.addEventListener(eventName, listener))
20 }
21
22 const captureViewportEvent = eventInfo => console.log(eventInfo)
23
24 const browser = await puppeteer.launch({headless: false,
25                                         defaultViewport: null})
26
27 browser.on('targetcreated', async target => {
28   if(target.type() === 'page') {
29     const page = await target.page()
30     await connectEvents(page)
31   }
32 })
33
34 const startingPage = (await browser.pages())[0]
35 await connectEvents(startingPage)
```

Ukázka kódu 24: Zachytávání událostí z viewportu

Puppeteer vždy spouští Chrome s jedním již otevřeným tabem, vzhledem k tomu,

že se nejedná o explicitně otevřený tab, jeho otevření nezpůsobí vykonání ❹. To je důvodem, proč navíc voláme ❶ na posledním řádku. Všechny ostatní vytvořené taby mají inicializované hlášení událostí pomocí kódu v ❺.

Napojení JavaScriptu Chromu a JavaScriptu, ve kterém běží Puppeteer se provede ve ❸. Druhý parametr udává metodu ke zveřejnění do window Chromu, první parametr je jméno uvnitř okna prohlížeče, které metodě odpovídá.

V ❺ se vypisují ohlášené události ze ❹. O provedení těla ❹ z JavaScriptu prohlížeče bychom přišli při navigaci, pokud bychom nezavolali ❷ při inicializaci hlášení.

A.2 Momentálně aktivní tab

```
1 async function getActivePage(browser) {
2   const pages = await browser.pages();
3   for (const p of pages) {
4     const isVisible = await p.evaluate(() => {
5       return document.visibilityState === 'visible'
6     })
7     if(isVisible)
8       return p
9   }
10  throw new Error('No page is currently active.')
11 }
```

Ukázka kódu 25: Metoda, která vrátí momentálně aktivní tab

A.3 Oprávnění „push“

```
1 import puppeteer from 'puppeteer'
2
3 (async () => {
4   const browser = await puppeteer.launch({headless: false})
5   const context = browser.defaultBrowserContext()
6   await context.overridePermissions('https://seznam.cz', ['push'])
7   await browser.close()
8 }) ()
```

Ukázka kódu 26: Pokus o získání oprávnění „push“

Výjimka se objeví na 6. řádku.

A.4 Element podle lokátoru

```
1  async function elementByAttribute(page, attrName, attrValue) {
2      const xpath = `//*[@${attrName} = '${attrValue}']`
3      const found = await page.$x(xpath)
4      if(found.length === 1)
5          return found[0]
6      else if (found.length === 0)
7          throw new Error(`Element not found: ${xpath}`)
8      else {
9          console.log(`Warning Xpath: ${xpath} is ambiguous`)
10         return found[0]
11     }
12 }
13
14 async function elementByLinkText(page, linkText) {
15     const xpath = `//*[@text() = '${linkText}']`
16     const found = await page.$x(xpath)
17     if(found.length === 1)
18         return found[0]
19     else if (found.length === 0)
20         throw new Error(`Element not found: ${xpath}`)
21     else {
22         console.log(`Warning Xpath: ${xpath} is ambiguous`)
23         return found[0]
24     }
25 }
```

Ukázka kódu 27: Metody převádějící vybrané lokátory na XPath s návratovou hodnotu odpovídajících elementů

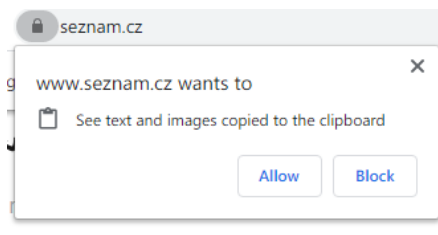
A.5 Práce se stránkou

```
1 (async () => {
2   const browser = await puppeteer.launch({headless: false})
3   const pages = await browser.pages()
4   let page0 = pages[0]
5
6   await Promise.all([
7     page0.waitForNavigation(),
8     page0.goto('https://www.seznam.cz/')
9   ])
10
11   await page0.waitForXPath('//*[@name = 'username']')
12   await elementByAttribute(page0, 'name', 'username')
13     .then(el => el.type('uzivatel', { "delay": 100 }))
14
15
16   await elementByAttribute(page0, 'name', 'username')
17     .then(el => el.evaluate((el) => {
18       el.setSelectionRange(0, 5, 'backward') // 'uziv'
19     })))
20
21   ❶ await page0.evaluate(async () => {
22     const currSelection = window.getSelection().toString()
23     await navigator.clipboard.writeText(currSelection)
24   })
25
26   await page0.waitForSelector('.search-form__input')
27   await page0.click('.search-form__input')
28
29   await page0.$eval('.search-form__input', async el => {
30     el.value = await navigator.clipboard.readText()
31   })
32 }) ()
```

Ukázka kódu 28: Pokus o zápis a čtení ze stránky bez jakéhokoliv oprávnění

Kód z uk. k. 28 je nutné doplnit o metodu `elementByAttribute`, jejíž implementace je uvedena v A.4.

Zatímco zápis do stránky proběhne úspěšně, při čtení se zobrazí výzva k udělení/zamítnutí oprávnění.



Obrázek 52: Výzva pro udělení/zamítnutí oprávnění pro čtení schránky

A.6 Pokus o získání oprávnění pro práci se schránkou

Zkusíme-li rozšířit uk. k. 28 z A.5 přidáním níže uvedené uk. k. 29 za 2. řádek, program se ukončí na řádce označeném ⑦ (uk. k. 28) a ohlásí výjimku „Error: Evaluation failed: DOMException: Write permission denied.“.

```
1 const ctx = browser.defaultBrowserContext()
2 await ctx.overridePermissions(
3   'https://www.seznam.cz',
4   ['clipboard-read', 'clipboard-write']
5 )
```

Ukázka kódu 29: Pokus o získání oprávnění pro čtení a zápis schránky (nepřímý způsob)

Stejný dopad bude mít i následující rozšíření, které je ale nutné přidat až za 4. řádek.

```
1 const cdpClient = await page0.target().createCDPSession()
2 await cdpClient.send('Browser.grantPermissions', {
3   origin: 'https://www.seznam.cz',
4   permissions: ['clipboardReadWrite']
5 })
```

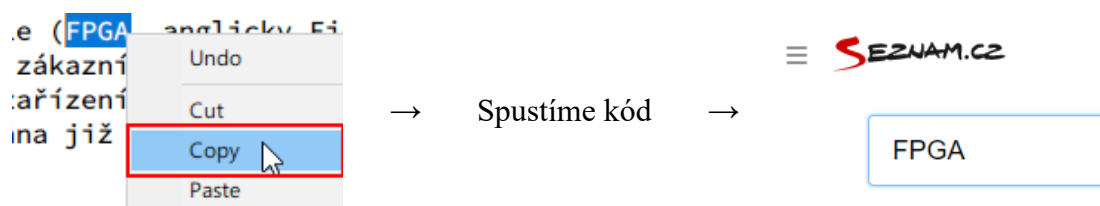
Ukázka kódu 30: Pokus o získání oprávnění pro čtení a zápis schránky (nativní způsob)

A.7 Čtení ze schránky s CDP oprávněním „clipboardReadWrite“

```
1 (async () => {
2   const browser = await puppeteer.launch({headless: false})
3
4   /*
5    const ctx = browser.defaultBrowserContext()
6    await ctx.overridePermissions(
7      'https://www.seznam.cz',
8      ['clipboard-read', 'clipboard-write']
9    )
10  */
11
12  const pages = await browser.pages()
13  let page0 = pages[0]
14
15  /* Tento úsek kódu je ekvivalentní s~výše zakomentovaným
16     fragmentem */
17  const cdpClient = await page0.target().createCDPSession()
18  await cdpClient.send('Browser.grantPermissions', {
19    origin: 'https://www.seznam.cz',
20    permissions: ['clipboardReadWrite']
21  })
22
23  await Promise.all([
24    page0.waitForNavigation(),
25    page0.goto('https://www.seznam.cz/')
26  ])
27
28  await page0.waitForSelector('.search-form__input')
29  await page0.click('.search-form__input')
30
31  await page0.$eval('.search-form__input', async el => {
32    el.value = await navigator.clipboard.readText()
33  })
34 })()
```

Ukázka kódu 31: Test čtení ze schránky s CDP oprávněním „clipboardReadWrite“

Výše uvedený kód můžeme využít k ověření funkčnosti čtení ze schránky takto:



Obrázek 53: Využití uk. k. 31 k ověření funkčnosti čtení ze schránky

A.8 Úspěšné čtení ze schránky i zápis do schránky

Fungující příklad vyrobíme rozšířením uk. k. 28 z A.5 přidáním uk. k. 32 za 4. řádek.

```
1  const cdpClient = await page0.target().createCDPSession()
2  await cdpClient.send('Browser.grantPermissions', {
3    origin: 'https://www.seznam.cz',
4    permissions: ['clipboardReadWrite', 'clipboardSanitizedWrite']
5  })
```

Ukázka kódu 32: Získání oprávnění „clipboardReadWrite“
a „clipboardSanitizedWrite“

A.9 Využití NetMQ a ZeroMQ.js

Následují dvě ukázky, jedna pro C#, druhá pro JavaScript s Node.js. Při spuštění obou ukázek se navzájem pošle verze běhového prostředí. Povšimněme si, že nepotřebujeme message broker, a navíc můžeme odesílat přímo textové řetězce.

```
1  using System;
2  using NetMQ;
3  using NetMQ.Sockets;
4
5  class Program {
6      static void Main(string[] args) {
7          using (PairSocket pair =
8              new PairSocket("@tcp://127.0.0.1:3333")) {
9
10             pair.SendFrame(".NET Framework " + typeof(string).Assembly.
11                 ImageRuntimeVersion);
12
13             string nodejsVersion = pair.ReceiveFrameString();
14             Console.WriteLine(nodejsVersion);
15         }
16     }
```

(a) C#

```
1 import zmq from 'zeromq'
2
3 let sock = new zmq.Pair
4 sock.connect('tcp://127.0.0.1:3333')
5
6 await sock.send('Node.js ' + process.version)
7 const dotnetVersion = (await sock.receive()).toString('utf-8')
8 console.log(dotnetVersion)
```

(b) JavaScript

Ukázka kódu 33: Demo zobrazující použití NetMQ a ZeroMQ.js

Slovníček vybraných pojmů

Automatizace prohlížeče Proces bezobslužného ovládání prohlížeče.

Chrome DevTools Protocol Protokol umožňující externím nástrojům ovládat a debugovat Chrome [9].

DevTools (Developer Tools) Sada nástrojů vestavěná ve Chromu umožňující debugování a diagnostiku stránek [8].

Electron Framework pro vytváření nativních aplikací pro OS Windows/macOS/Linux pomocí JavaScriptu, HTML, CSS a dalších webových technologií [11].

Extension API (Extension Application Programming Interface) Rozhraní pro programování rozšíření do Chromu [7,47].

Firefox Remote Protocol Protokol využívaný ve Firefoxu kompatibilní s CDP. Implementována je pouze podmnožina CDP [36,37].

Headless Režim spuštění prohlížeče bez grafického uživatelského rozhraní [5].

Headless Recorder (dříve Puppeteer Recorder) Nahrávač pro Puppeteer implementovaný s využitím Extension API [16].

Identifikátor elementu Libovolný způsob (selektor, lokátor, XPath, ...) určení elementu na stránce.

Katalon Recorder Konkurenční řešení pro nahrávání a přehrávání akcí. Fork Selenia IDE [20].

Kontext prohlížeče Prostředí jedné i více stránek prohlížeče. Obvykle se sdíleným nastavením, např. neukládat historii [6,43].

Lokátor Rozšířený identifikátor, umožňující lépe identifikovat elementy, zejména textovým popiskem elementu. Platí, že lokátory \supset selektory [44].

Message Broker Samostatný program zpracovávající zprávy od odesílatele, které předá adresátovi. Často využívaný pokud obě strany nepoužívají stejný poštovní protokol [76].

Mono Implementace .NET Framework s otevřeným zdrojovým kódem podporující platformy: Linux, macOS, BSD a Windows.

Nahrávač (recorder) Nástroj pro nahrávání.

Nahrávání (recording) Proces zaznamenávání běžných uživatelských akcí prováděných s prohlížečem.

Navigace Načtení jiné stránky s odlišnou URL [31].

Node.js Běhové prostředí JavaScriptu běžící vně prohlížeče [3].

PhantomJS Headless prohlížeč speciálně navržený tak, aby byl skriptovatelný [26].

Playwright Fork Puppeteeru spravovaná firmou Microsoft [29].

Puppeteer Knihovna pro Node.js, která poskytuje API pro ovládání Chromu využitím CDP. Ve výchozí konfiguraci spouští prohlížeč v headless režimu [31].

Přehrávač (player) Nástroj pro přehrávání.

Přehrávání (playing) Proces automatického vykonávání nahraných akcí v prohlížeči.

Ranorex Recorder Komerční řešení pro automatizaci GUI aplikací a prohlížečů [49].

Selenium Konkurenční sada technologií pro ovládání prohlížečů [2].

Selenium IDE Konkurenční nahrávač a přehrávač pro Selenium [42].

TypeScript Rozšíření JavaScriptu o typy a jejich statickou kontrolu. TypeScript je vyvinutý a spravovaný firmou Microsoft [55].

Viewport Viditelná část stránky prohlížeče [46].

Seznam obrázků

1	Komunikace Selenia s prohlížečem	6
2	Komunikace Puppeteeru s Chromem	8
3	Výběr z lokátorů identifikujících element	16
4	Změna elementu bez ručního přepisování lokátoru	17
5	Zpětné zobrazení identifikovaného elementu	17
6	Stránky MFF UK s nabídkovým elementem	18
7	Explicitní zaznamenání „mouseover“ akce	18
8	Přehrávání s breakpointem na 3. akci	19
9	Porovnání GUI Katalon Recorderu a Selenia IDE	20
10	Porovnání podporovaných jazyků pro export	21
11	Použití Headless Recorderu	23
12	Nastavení Headless Recorderu	24
13	Několik elementů, jejichž změna hodnoty vyvolá událost „change“ . .	30
14	Diagram řešení	35
15	Diagram hlavního okna	36
16	Diagram editace existující nahrávky	37
17	Diagram editace nastavení	38
18	Nastavení připojení ke Chromu	44
19	Nastavení cesty k interpreteru Node.js a backendu	45
20	Vytvoření nové nahrávky	45
21	Editací UI pro nahrávky	46
22	Změna jména nahrávky	46
23	Spouštění backendu a Chromu	46
24	Editací UI po spuštění backendu a prohlížeče	47
25	Zapnutí nahrávání akcí	47
26	Nahrání první akce	47
27	Vypnutí nahrávání akcí	48
28	Optimalizace akcí	48
29	Zobrazení kódu pro Puppeteer odpovídající nahraným akcím	49
30	Zapnutí přehrávání akcí	49
31	Spuštění přehrávání akcí	50
32	Okno popisující stav spuštěného přehrávání	50
33	Zvýraznění akce, která vyvolala chybu	51
34	Odpojení prohlížeče a vypnutí backendu	51
35	Uložení nahrávky a návrat do seznamu všech uložených nahrávek . .	52
36	Schéma zapojení počítačů	52
37	Nastavení připojení na „Connect“ s vyplněním IP adresy a portu . . .	53
38	Tlačítko JSON jako součást každé akce	54

39	Data ve formátu JSON popisující nahranou akci „pageUrlChanged“	54
40	Modifikace nezpracovaných dat v JSONu	55
41	Přepínače pro výběr akcí ke zpracování	55
42	Zaškrťovací políčka pro výběr a povolení akcí	56
43	Vygenerování kódu jedné akce typu „pageUrlChanged“	56
44	Otevření okna pro aplikaci filtrů	57
45	Aktivace filtru pro akce pocházející z okna prohlížeče a viewportu	58
46	Skrytí dvou akcí typu „click“ pomocí filtru	59
47	Vztahy mezi přepínači a políčky	60
48	Seznam dostupných lokátorů pro konkrétní akci typu „click“	60
49	Vlastní lokátor	61
50	Data ve formátu JSON akce typu „click“	61
51	Přidání nového lokátoru	62
52	Výzva pro udělení/zamítnutí oprávnění pro čtení schránky	68
53	Využití uk. k. 31 k ověření funkčnosti čtení ze schránky	69

Seznam tabulek

1	Využitý software a technologie včetně verzí	4
2	Seznam lokátorů podporovaných Seleniem [44]	7
3	Podporované prohlížeče Seleniem [44]	7
4	Několik odpovídajících verzí Chromu a Puppeteeru [30]	8
5	Porovnání čekání Playwrightu a Puppeteeru [27, 30]	12
6	Porovnání datumu vydání prvních verzí prohlížečů podporujících headless režim [12, 18, 32]	13
7	Orientační porovnání existujících řešení	15
8	Oprávnění pro schránku definované CDP a Puppeteerem	32
9	Skutečný význam oprávnění CDP pro schránku	32
10	Porovnání podporovaných událostí viewportu napříč řešeními	39
11	Porovnání podporovaných událostí okna prohlížeče napříč řešeními	40
12	Porovnání podporovaných identifikátorů elementů napříč řešeními	40
13	Porovnání podporovaných druhů čekání napříč řešeními	41
14	Porovnání podpory generování kódu napříč řešeními	42
15	Porovnání dalších pokročilých funkcí napříč řešeními	43

Seznam ukázek kódu

1	HTML kód tlačítka pro přihlášení	6
2	Screenshot v Node.js kódu	9
3	Node.js kód pro vyhodnocení JavaScriptu prohlížeče	10

4	Node.js kód pro kliknutí na element	10
5	Porovnání kódu Playwrightu a Puppeteeru pro stisk elementu po navigaci	11
6	Akce se specificky identifikovanými elementy	12
7	XPath ekvivalentní identifikátoru elementu na 4. řádku	12
8	Stažení instalátoru Firefoxu	13
9	Uložení screenshotu celé stránky do PDF s využitím PhantomJS	14
10	Zachycení otevření nového tabu	25
11	Otevření nového tabu	25
12	Zachycení uzavření existujícího tabu	26
13	Uzavření existujícího tabu	26
14	Zachycení změny adresy URL tabu	26
15	Nastavení tabu jako aktivního	27
16	Kliknutí na element odpovídající selektoru	28
17	Přejetí myši na element odpovídající selektoru	28
18	Odeslání formuláře pomocí Web API	28
19	Rolování stránky	29
20	Dvojitě kliknutí na element odpovídající selektoru	29
21	Výběr textu	30
22	Přehrávání události „change“	30
23	Přesměrování vzdáleně přístupného portu 9223 na port 9222	53
24	Zachytávání událostí z viewportu	64
25	Metoda, která vrátí momentálně aktivní tab	65
26	Pokus o získání oprávnění „push“	65
27	Metody převádějící vybrané lokátory na XPath s návratovou hodnotu odpovídajících elementů	66
28	Pokus o zápis a čtení ze schránky bez jakéhokoliv oprávnění	67
29	Pokus o získání oprávnění pro čtení a zápis schránky (nepřímý způsob)	68
30	Pokus o získání oprávnění pro čtení a zápis schránky (nativní způsob)	68
31	Test čtení ze schránky s CDP oprávněním „clipboardReadWrite“	69
32	Získání oprávnění „clipboardReadWrite“ a „clipboardSanitizedWrite“	70
33	Demo zobrazující použití NetMQ a ZeroMQ.js	71

Reference

- [1] 16 reasons why to use Selenium IDE in 2019 (and 2 why not) - Automated Visual Testing | Applitools. Automated Visual Testing with Visual AI [online]. Copyright © 2020 Applitools. All Rights Reserved. [cit. 14.11.2020]. Dostupné z: <https://applitools.com/blog/why-selenium-ide-2019>
- [2] About Selenium. SeleniumHQ Browser Automation [online] [cit. 2.5.2020]. Dostupné z: <https://www.selenium.dev/about>
- [3] About | Node.js. [online]. Copyright © OpenJS Foundation. All Rights Reserved. Portions of this site originally [cit. 30.10.2020]. Dostupné z: <https://nodejs.org/en/about>
- [4] Archiving the project: suspending the development · Issue #15344 · ariya/phantomjs · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 7.11.2020]. Dostupné z: <https://github.com/ariya/phantomjs/issues/15344>
- [5] Automated testing with Headless Chrome | Web | Google Developers. Google Developers [online] [cit. 2.5.2020]. Dostupné z: <https://developers.google.com/web/updates/2017/06/headless-karma-mocha-chai>
- [6] Browsing context - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. [online]. Copyright © 2005 [cit. 7.11.2020]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Browsing_context
- [7] Chrome APIs - Google Chrome. [online] [cit. 2.5.2020]. Dostupné z: https://developer.chrome.com/extensions/api_index
- [8] Chrome DevTools | Google Developers. Google Developers [online] [cit. 2.5.2020]. Dostupné z: <https://developers.google.com/web/tools/chrome-devtools>
- [9] Chrome DevTools Protocol [online] [cit. 2.11.2020]. Dostupné z: <https://chromedevtools.github.io/devtools-protocol>
- [10] Ecosystem. SeleniumHQ Browser Automation [online] [cit. 31.10.2020]. Dostupné z: <https://www.selenium.dev/ecosystem>
- [11] Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS. [online] [cit. 14.11.2020]. Dostupné z: <https://www.electronjs.org>
- [12] Firefox Release Calendar - MozillaWiki. [online] [cit. 8.11.2020]. Dostupné z: https://wiki.mozilla.org/Release_Management/Calendar
- [13] Frequently asked questions - Firefox add-on technology is modernizing | Firefox Help [online]. Copyright ©1998 [cit. 14.11.2020]. Dostupné z: <https://support.mozilla.org/en-US/kb/frequently-asked-questions-firefox-addon>

- [14] Getting Started with Headless Chrome | Web | Google Developers. Google Developers [online] [cit. 7.11.2020]. Dostupné z: <https://developers.google.com/web/updates/2017/04/headless-chrome>
- [15] GitHub - SeleniumHQ/selenium-ide at v3. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 14.11.2020]. Dostupné z: <https://github.com/SeleniumHQ/selenium-ide/tree/v3>
- [16] GitHub - checkly/headless-recorder: Headless recorder is a Chrome extension that records your browser interactions and generates a Puppeteer or Playwright script.. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 30.10.2020]. Dostupné z: <https://github.com/checkly/headless-recorder>
- [17] GitHub - microsoft/playwright: Node.js library to automate Chromium, Firefox and WebKit with a single API. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 5.11.2020]. Dostupné z: <https://github.com/microsoft/playwright>
- [18] Google Chrome version history - Wikipedia. [online] [cit. 8.11.2020]. Dostupné z: https://en.wikipedia.org/wiki/Google_Chrome_version_history
- [19] Index. SeleniumHQ Browser Automation [online] [cit. 3.11.2020]. Dostupné z: <https://www.selenium.dev/selenium/docs/api/javascript>
- [20] Katalon Recorder (Selenium tests generator) - Internetový obchod Chrome. [online] [cit. 2.5.2020]. Dostupné z: <https://chrome.google.com/webstore/detail/katalon-recorder-selenium/ljdobmomdgdlniojadhoplhkpialdid>
- [21] Katalon Recorder vs Katalon Studio | Katalon Docs. Redirecting... [online]. Copyright © 2020 Katalon LLC. All rights reserved. [cit. 17.11.2020]. Dostupné z: <https://docs.katalon.com/katalon-recorder/docs/katalon-recorder-vs-katalon-studio.html>
- [22] Katalon Solution. Katalon Solution [online]. Copyright © 2020 Katalon LLC. All rights reserved. [cit. 16.11.2020]. Dostupné z: <https://www.katalon.com/katalon-recorder-ide>
- [23] Katalon Solution. Katalon Solution [online]. Copyright © 2020 Katalon LLC. All rights reserved. [cit. 16.11.2020]. Dostupné z: <https://www.katalon.com/pricing>
- [24] Katalon Studio - Wikipedia. [online] [cit. 16.11.2020]. Dostupné z: https://en.wikipedia.org/wiki/Katalon_Studio
- [25] NavigatorID.appName - Web APIs | MDN. [online]. Copyright © 2005 [cit. 3.11.2020]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/NavigatorID/appName>

- [26] PhantomJS - Scriptable Headless Browser. PhantomJS - Scriptable Headless Browser [online]. Copyright © 2010 [cit. 7.11.2020]. Dostupné z: <https://phantomjs.org>
- [27] playwright/api.md at master · microsoft/playwright · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 5.11.2020]. Dostupné z: <https://github.com/microsoft/playwright/blob/master/docs/api.md>
- [28] playwright/browser_patches at master · microsoft/playwright · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 5.11.2020]. Dostupné z: https://github.com/microsoft/playwright/tree/master/browser_patches
- [29] Playwright. Playwright [online] [cit. 5.11.2020]. Dostupné z: <https://playwright.dev>
- [30] puppeteer/api.md at main · puppeteer/puppeteer · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 3.11.2020]. Dostupné z: <https://github.com/puppeteer/puppeteer/blob/main/docs/api.md>
- [31] Puppeteer [online] [cit. 2.11.2020]. Dostupné z: <https://pptr.dev>
- [32] Release 1.0.0 · ariya/phantomjs · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 7.11.2020]. Dostupné z: <https://github.com/ariya/phantomjs/releases/tag/1.0.0>
- [33] Releases · SeleniumHQ/selenium-ide · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 14.11.2020]. Dostupné z: <https://github.com/SeleniumHQ/selenium-ide/releases>
- [34] Releases · microsoft/playwright · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 5.11.2020]. Dostupné z: <https://github.com/microsoft/playwright/releases?after=v0.13.0>
- [35] Releases · puppeteer/puppeteer · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 2.11.2020]. Dostupné z: <https://github.com/puppeteer/puppeteer/releases?after=v0.12.0>
- [36] Remote - MozillaWiki. [online] [cit. 2.11.2020]. Dostupné z: <https://wiki.mozilla.org/Remote>
- [37] Remote Protocol — Firefox Source Docs documentation. Firefox Source Tree Documentation — Firefox Source Docs documentation [online] [cit. 2.11.2020]. Dostupné z: <https://firefox-source-docs.mozilla.org/remote/index.html>

- [38] Selectors Level 3. World Wide Web Consortium (W3C) [online]. Copyright © 2018 [cit. 9.11.2020]. Dostupné z: <https://www.w3.org/TR/selectors-3>
- [39] Selectors Level 4. World Wide Web Consortium (W3C) [online]. Copyright © 2018 [cit. 9.11.2020]. Dostupné z: <https://www.w3.org/TR/selectors-4>
- [40] Selenium (software) - Wikipedia. [online] [cit. 2.11.2020]. Dostupné z: [https://en.wikipedia.org/wiki/Selenium_\(software\)](https://en.wikipedia.org/wiki/Selenium_(software))
- [41] Selenium History. SeleniumHQ Browser Automation [online] [cit. 14.11.2020]. Dostupné z: <https://www.selenium.dev/history>
- [42] Selenium IDE · Open source record and playback test automation for the web. SeleniumHQ Browser Automation [online]. Copyright © 2019 Software Freedom Conservancy [cit. 30.10.2020]. Dostupné z: <https://www.selenium.dev/selenium-ide>
- [43] Terminology - HTML5. W3C on GitHub [online] [cit. 7.11.2020]. Dostupné z: <https://w3c.github.io/html-reference/terminology.html>
- [44] The Selenium Browser Automation Project :: Documentation for Selenium [online] [cit. 2.11.2020]. Dostupné z: <https://www.selenium.dev/documentation/en>
- [45] Using Headless Mode in Firefox - Mozilla Hacks - the Web developer blog. Home - Mozilla Hacks - the Web developer blog [online] [cit. 7.11.2020]. Dostupné z: <https://hacks.mozilla.org/2017/12/using-headless-mode-in-firefox>
- [46] Viewport concepts - CSS: Cascading Style Sheets | MDN. [online]. Copyright © 2005 [cit. 7.5.2020]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/Viewport_concepts
- [47] What are extensions? - Google Chrome. [online] [cit. 2.5.2020]. Dostupné z: <https://developer.chrome.com/extensions>
- [48] Window: load event - Web APIs | MDN. [online]. Copyright © 2005 [cit. 5.11.2020]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event
- [49] Company Profile | About Ranorex. Test Automation for GUI Testing | Ranorex [online]. Copyright © 2020 Ranorex GmbH. All Rights Reserved [cit. 9.12.2020]. Dostupné z: <https://www.ranorex.com/company>
- [50] Request: browser.currentPage() or similar way to access Pages · Issue #443 · puppeteer/puppeteer · GitHub. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 12.12.2020]. Dostupné z: <https://github.com/puppeteer/puppeteer/issues/443>

- [51] `HTMLFormElement.submit()` - Web APIs | MDN. [online]. Copyright © 2005 [cit. 13.12.2020]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/HTMLFormElement/submit>
- [52] `Window` - Web APIs | MDN. [online]. Copyright © 2005 [cit. 13.12.2020]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/Window>
- [53] `Element` - Web APIs | MDN. [online]. Copyright © 2005 [cit. 13.12.2020]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/Element>
- [54] `HTMLElement` - Web APIs | MDN. [online]. Copyright © 2005 [cit. 13.12.2020]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement>
- [55] TypeScript: Typed JavaScript at Any Scale.. TypeScript: Typed JavaScript at Any Scale. [online]. Copyright © 2012 [cit. 17.12.2020]. Dostupné z: <https://www.typescriptlang.org>
- [56] The history of C# - C# Guide | Microsoft Docs. [online]. Copyright © Microsoft 2020 [cit. 18.12.2020]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history>
- [57] Home | Mono. Home | Mono [online]. Copyright © 2020 Mono Project [cit. 18.12.2020]. Dostupné z: <https://www.mono-project.com>
- [58] Visual Studio IDE, Code Editor, Azure DevOps, & App Center - Visual Studio. [online] [cit. 18.12.2020]. Dostupné z: <https://visualstudio.microsoft.com>
- [59] winforms - Windows Forms Dead. Long life to WPF - Stack Overflow. Stack Overflow - Where Developers Learn, Share, & Build Careers [online] [cit. 18.12.2020]. Dostupné z: <https://stackoverflow.com/questions/2632118/windows-forms-dead-long-life-to-wpf>
- [60] winforms - Will Windows Forms be deprecated in favor of WPF? - Stack Overflow. Stack Overflow - Where Developers Learn, Share, & Build Careers [online] [cit. 18.12.2020]. Dostupné z: <https://stackoverflow.com/questions/913417/will-windows-forms-be-deprecated-in-favor-of-wpf>
- [61] wpf - Are Windows Forms old tech? - Stack Overflow. Stack Overflow - Where Developers Learn, Share, & Build Careers [online] [cit. 18.12.2020]. Dostupné z: <https://stackoverflow.com/questions/1916510/are-windows-forms-old-tech>
- [62] Is WPF replacement of WinForms? - Stack Overflow. Stack Overflow - Where Developers Learn, Share, & Build Careers [online] [cit. 18.12.2020]. Dostupné z: <https://stackoverflow.com/questions/4852407/is-wpf-replacement-of-winforms/4852581>

- [63] .net - When creating a new GUI, is WPF the preferred choice over Windows Forms? - Stack Overflow. Stack Overflow - Where Developers Learn, Share, & Build Careers [online] [cit. 18.12.2020]. Dostupné z: <https://stackoverflow.com/questions/57909/when-creating-a-new-gui-is-wpf-the-preferred-choice-over-windows-forms>
- [64] What is WPF? - Visual Studio | Microsoft Docs. [online]. Copyright © Microsoft 2020 [cit. 19.12.2020]. Dostupné z: <https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2019>
- [65] UWP Documentation - UWP app developer - UWP applications | Microsoft Docs. [online]. Copyright © Microsoft 2020 [cit. 19.12.2020]. Dostupné z: <https://docs.microsoft.com/en-us/windows/uwp>
- [66] Compatibility | Mono. Home | Mono [online]. Copyright © 2020 Mono Project [cit. 19.12.2020]. Dostupné z: <https://www.mono-project.com/docs/about-mono/compatibility>
- [67] Choose your Windows app platform - Windows applications | Microsoft Docs. [online]. Copyright © Microsoft 2020 [cit. 19.12.2020]. Dostupné z: <https://docs.microsoft.com/en-us/windows/apps/desktop/choose-your-platform>
- [68] Index | Node.js v15.4.0 Documentation. [online] [cit. 19.12.2020]. Dostupné z: <https://nodejs.org/api>
- [69] .NET API browser | Microsoft Docs. [online]. Copyright © Microsoft 2020 [cit. 19.12.2020]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/api/?view=netframework-4.8>
- [70] [online]. Copyright © 2020 The ZeroMQ authors [cit. 19.12.2020]. Dostupné z: <https://zeromq.org>
- [71] GitHub - zeromq/clrzmq4: ZeroMQ C# namespace (.NET and mono, Windows, Linux and MacOSX, x86 and amd64). GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 19.12.2020]. Dostupné z: <https://github.com/zeromq/clrzmq4>
- [72] [online]. Copyright © 2020 The ZeroMQ authors [cit. 19.12.2020]. Dostupné z: <https://zeromq.org/languages/csharp>
- [73] GitHub - zeromq/netmq: A 100% native C# implementation of ZeroMQ for .NET. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 19.12.2020]. Dostupné z: <https://github.com/zeromq/netmq>
- [74] GitHub - zeromq/zeromq.js: Node.js bindings to the ØMQ library. GitHub: Where the world builds software · GitHub [online]. Copyright © 2020 GitHub, Inc. [cit. 19.12.2020]. Dostupné z: <https://github.com/zeromq/zeromq.js>

- [75] [online]. Copyright © 2020 The ZeroMQ authors [cit. 19.12.2020]. Dostupné z: <https://zeromq.org/get-started>
- [76] What are Message Brokers? | IBM. [online]. Copyright © Copyright IBM Corporation 2020 [cit. 28.12.2020]. Dostupné z: <https://www.ibm.com/cloud/learn/message-brokers>
- [77] google chrome - Using Chromium Remote Debugging from External Device - Stack Overflow. Stack Overflow - Where Developers Learn, Share, & Build Careers [online] [cit. 4.1.2021]. Dostupné z: <https://stackoverflow.com/questions/18506233/using-chromium-remote-debugging-from-external-device>